平成 29 年度　修士論文

低遅延マルチメディア処理のための
仮想クラウド基盤を活用した
エッジクラウドシステムに関する検討

早稲田大学　基幹理工学研究科　情報理工・情報通信専攻

5116F013-5

今金　健太郎

指導　甲藤二郎 教授

2018 年 1 月 30 日

指導教授印	受付印

目次

第 1 章 序論

1.1 はじめに

スマートフォンや Internet of Things (IoT) デバイスといった端末の普及に伴い，アプリケーションサービスの高度・複雑化が進んでいる．これらを端末のみで実行することは，端末の消費電力増加や処理能力に対する懸念から現実的ではなく，クラウドコンピューティングを利用して，端末から遠隔地のデータセンタに存在するリソースでアプリケーション実行を行うことが主流となっている．しかし，アプリケーションの多様化に伴い，それらの実行に必要なデータがビッグデータ化している．Cisco のホワイトペーパー[1]によると，全世界のモバイルデータトラフィックは 2021 までに 2016 年比約 7 倍に増加すると予測されており，通信遅延の増大によって，今後クラウドコンピューティングを活用したオフロードではリアルタイム処理の要求を必ずしも満足し続けるとは言えない．

一方で，近年，エッジルータ，無線通信基地局，アクセスポイント等といったネットワークのエッジ部に分散配置したリソースを利用してアプリケーション処理を実行するエッジコンピューティング[2]が提案されている．エッジコンピューティングでは，アプリケーション処理要求を行う端末とエッジ間の物理的な距離が，同端末とクラウド間の距離に比べて大幅に短縮される．さらに，複数エリアに分散配置されているエッジサーバで分散的にアプリケーション処理を実行できるため，サーバ 1 台あたりの処理負担も軽減される．そのため，上記で挙げたクラウドコンピューティングの問題点を改善することが期待されている．しかし，エッジコンピューティングでは，それぞれのリソースが小規模なデータセンタのような構造で分散的に存在するため，全体で見るとリソースの配置が複雑になってしまう．従って，エッジコンピューティングを効率的に利用するために，分散配置された計算リソースの中から，ユーザが要求する Quality of Service (QoS)を満たすように単一エリアのエッジサーバ内及び複数エリアのエッジサーバ間で使用するリソースを正しく選択する必要がある．

1.2 研究目的

本研究では，はじめに，従来のクラウドコンピューティングおよびエッジコンピューティング環境においてアプリケーション(本研究ではマルチメディア処理を対象とする)を実行する際の遅延特性をシナリオ別に理論モデル化し，それぞれの遅延分析を行うことでエッジコンピューティングの有効性を示す．さらに，エッジコンピューティング環境において，ネットワークの使用状況や計算資源を踏まえてさらなる低遅延処理を実現するための

ネットワーク経路の最適化を試みる．次に，エッジコンピューティングを活用し，ユーザが要求するアプリケーションを低遅延で実行するためのエッジクラウドシステムを提案する．本システムは，仮想サーバ・ネットワークという観点から効率的なリソース利用を実現するために OpenStack を活用しており，アプリケーション処理を「機能」レベルに分割し，システム内で機能や処理データも含め共有，再利用することで，並列分散処理によるアプリケーション処理の実行遅延の削減を図っている．本システムに対して，研究室内およびクラウドプロバイダが提供するサーバ上に評価環境を構築し，実機実験によるアプリケーション実行遅延評価を行うことで，システムの有効性を確認する．

1.3 本論文の構成

第 1 章では，本研究の目的について述べた．
第 2 章では，本研究に関連する技術について述べる．
第 3 章では，遅延分析モデルを用いたエッジコンピューティングにおけるマルチメディア処理実行遅延評価について述べる．
第 4 章では，仮想クラウド基盤を活用したエッジクラウドシステムにおけるマルチメディア処理実行遅延評価について述べる．
第 5 章では，本論文の総括を述べる．

第 2 章 関連技術

本章では，本研究で提案する低遅延マルチメディア処理のためのエッジクラウドシステムに関連する基本的事項として，コンピューティング資源の利用形態と仮想化技術，およびそれらが連携した仮想クラウドサービスの運用と構築について説明する．

2.1 コンピューティング資源の利用形態

2.1.1 クラウドコンピューティング

クラウドコンピューティング[3]は，遠隔地に存在するソフトウェアやハードウェア，ストレージ，メモリ等といったコンピューティング資源に対し，利用者がネットワークを通じてどこからでもアクセスすることを可能とする利用形態である．「クラウド」は，ユーザから見えない位置にあるコンピューティング資源，ネットワークのシステム図を雲(=cloud)のかたまりのように表現できることに由来する．クラウドコンピューティングの主なメリットとして，ネットワークを通じて利用者が慣れた仮想環境で場所を問わず利用できること,利用者の使用目的に合わせて 3 つのサービスおよび実装モデルが提供されていること，コンピュータ自体の管理が不要で，継続して資源を共用しながらシステムが稼動できること，必要なときに必要なだけリソースの追加やスケールアウトができることなどが挙げられる．一方，デメリットとしては，ネットワーク環境が必須であり，アクセス時に遅延が発生すること，セキュリティ面でリスクがある可能性があることが挙げられる．

ここで，メリットに挙げられていた 3 つのサービスモデルについてそれぞれ説明し，サービスモデルごとの準備・保守項目を図 2.1 に示す．

1) Software as a service (SaaS)
クラウドのインフラ上で稼動しているプロバイダ由来のアプリケーションが利用者に提供される．アプリケーションには，Web ブラウザのようなシンククライアント型のインターフェース，またはプログラムインターフェースを通じてアクセスし，サービスを利用することになる．利用者は，ネットワーク・ハードウェア・オペレーティングシステム・ミドルウェアの全ての要素について自身で準備・管理する必要がない．

2) Platform as a Service (PaaS)
クラウドのインフラ上にユーザが開発または購入したアプリケーションを実装する．アプリケーションはプロバイダがサポートするプログラミング言語やツールを用いて生み出

されるものであり，ユーザはネットワーク，オペレーティング，ストレージなど基盤のインフラを管理することはない．一方，自身の実装したアプリケーションとそのアプリケーションをホストする環境については，ユーザが管理を行う必要がある．

3) Infrastructure as a Service (IaaS)

ネットワーク・ハードウェア(CPU やメモリ，ハードディスク)その他コンピューティングリソースがユーザに提供され，オペレーティングシステムやアプリケーションを含む任意のソフトウェアを実装し，走らせることができる．また，PaaS 同様，ユーザは基盤のインフラを管理することはないが，オペレーティングシステムや，ストレージ，実装されたアプリケーションの管理を行う必要がある．

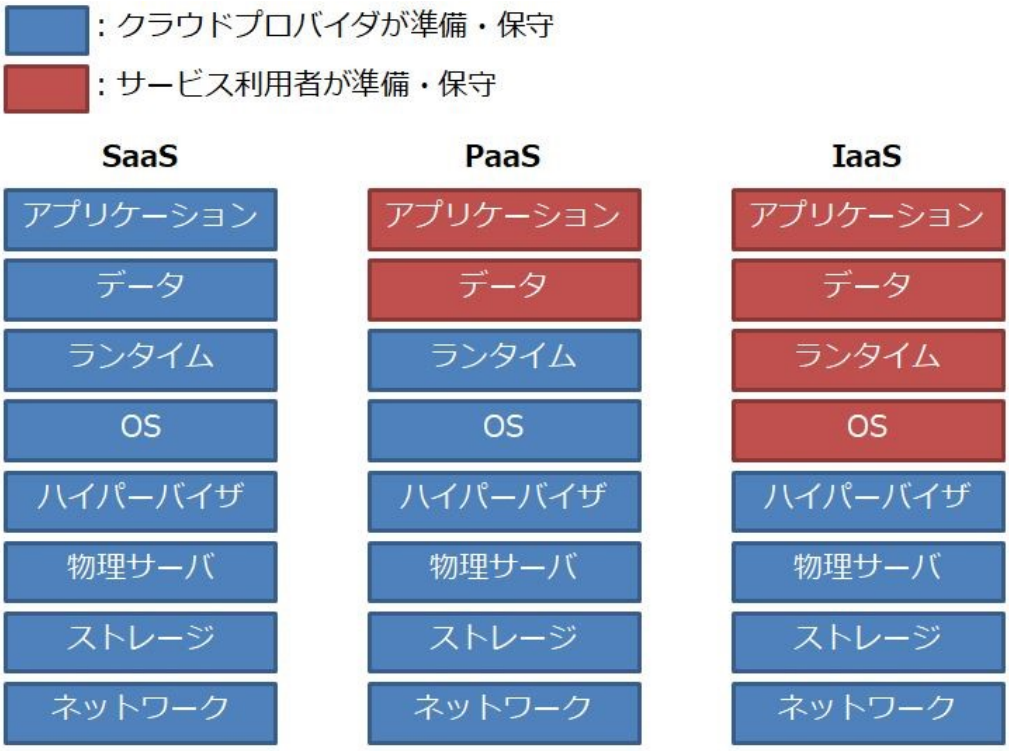

図 2.1 サービスモデルごとの準備・保守項目

上記サービスモデルと合わせて，メリットに挙げられていた 3 つの実装モデルについてそれぞれ説明し，それらの関連性を図 2.2 に示す．

1) Private Cloud

複数のユーザからなる特定の組織を対象として提供される形態のクラウド．その所有，管理および運用は，その組織自身または第三者により行われ，存在場所はその組織の施設内部または外部となる．

2) Public Cloud

不特定多数を対象として，広く一般の自由な利用に向けて提供される形態のクラウド．その所有，管理及び運用は企業組織や学術機関などにより行われ，存在場所はクラウドプロバイダの施設内となる．

3) Hybrid Cloud

2 つ以上の異なるクラウド(Private，Public 等)を組み合わせて利用する形態のクラウド．各クラウドは独立の存在であるが，標準化された，あるいは固有の技術で結合され，データとアプリケーションの移動可能性を実現している．

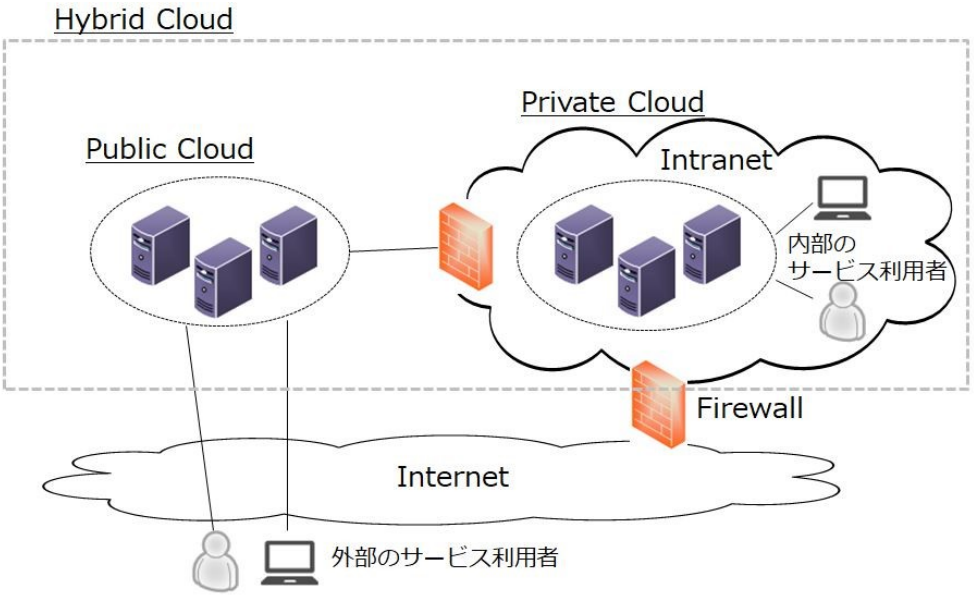

図 2.2 各実装モデルの関連性

2.1.2 エッジコンピューティング

エッジコンピューティング[2]では，エッジルータ，無線通信基地局，アクセスポイント等といったネットワークの端の部分（エッジ）に，ソフトウェアやデータ，ストレージなどのコンピューティングリソースを有したエッジサーバを分散配置する（図 2.3）．従来のクラウドコンピューティングでバックボーンのデータセンタに配置されるクラウドサーバは，ユーザ端末からのアクセス時に往復遅延時間（RTT: Round trip Time）が国内間で 100ms 未満，日米間 100ms 以上，日欧間 200ms 以上かかるのに対し，エッジサーバは，クラウドサーバに比べてユーザ端末からの物理的な距離が短縮されるため，数～数十 ms でアクセスできる．そのため，エッジコンピューティングを活用することで，従来のクラウドコン

ピューティングのボトルネックの一つである通信遅延を改善することができる．さらに，ユーザ端末，クラウドサーバ及びエッジサーバ間で高負荷なアプリケーションの分散処理を実行することで，端末の性能によらずに処理の高速化が可能となる．

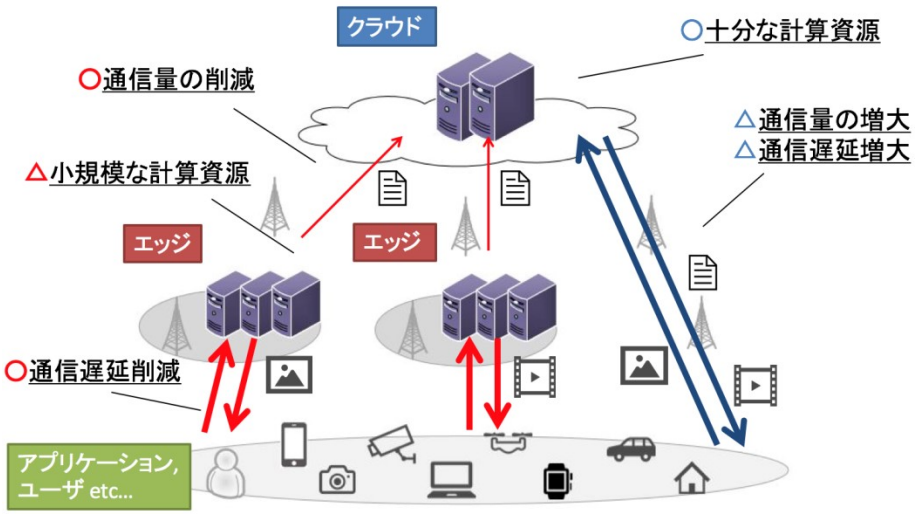

図 2.3 エッジコンピューティングと従来のクラウドコンピューティングの比較

ここで，クラウドコンピューティング，エッジコンピューティングのアプリケーション対象領域を図 2.4 に示す．エッジコンピューティングは分散環境になるため，クラウドコンピューティングで用いるような大容量のストレージと高い処理能力を持ったサーバを複数配置することは困難であり，おのずとクラウドコンピューティングとはアプリケーションの適用領域が異なってくる．そこで，[4]では，「ネットワークを使ったアプリケーションを，『リアルタイム性』と『サーバとの通信頻度』の 2 つの軸のマトリックスに当てはめていくと，クラウドとエッジコンピューティングの適用領域の違いが分かりやすい」と述べている．

クラウドコンピューティングには，リアルタイム性が低く，サーバとの通信頻度が低いアプリケーションが向いており，例として，検索，メール，Web サービス，オンラインストレージ，SNS 等が挙げられる．一方，エッジコンピューティングには，リアルタイム性が高く，サーバとの通信頻度が高いアプリケーションが向いており，高速道路交通システムと連携させた自動車制御，AR，センサー，監視カメラ，オンラインゲーム，測定器から大量の情報を集約する M2M などが挙げられる．これらのアプリケーションは，エッジサーバで一次処理を行い，その処理結果や注目すべき外れ値だけを中央の高性能なクラウドサーバに送ることで計算・通信資源の利用の局所化を実現し，ネットワーク基幹部分やクラウドサーバへのトラフィックの抑制が可能となっている．

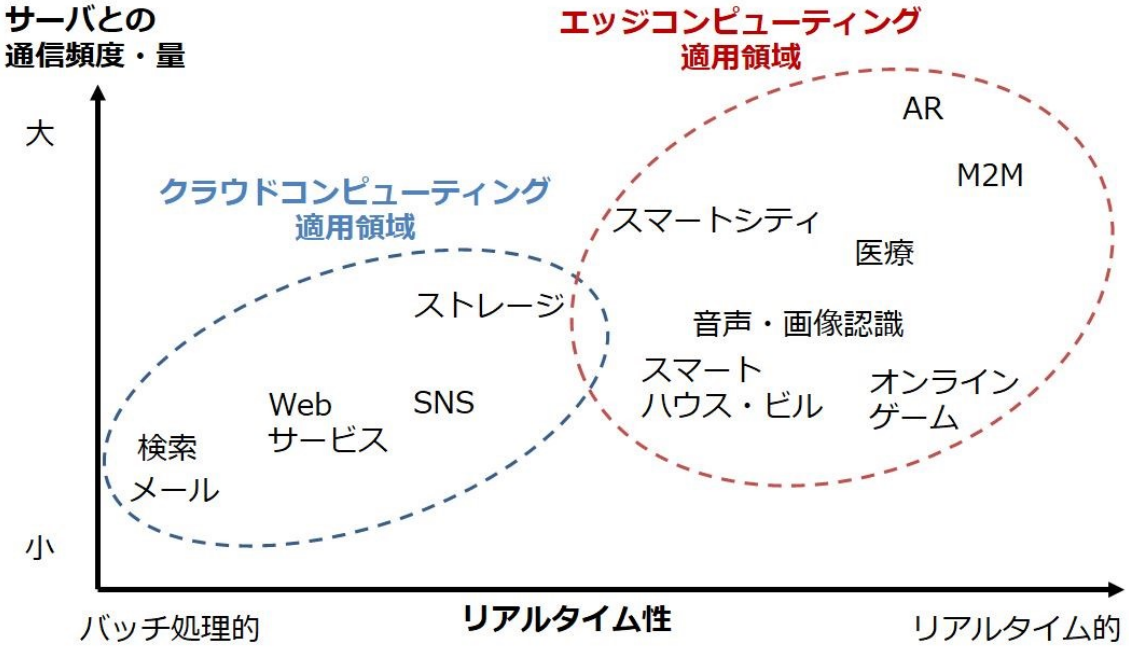

図 2.4 クラウド/エッジコンピューティングのアプリケーション適用領域

2.1.3 フォグコンピューティング

フォグコンピューティング[5]は, Cisco が提唱するコンピューティング資源の利用形態である. IoT デバイスのあるローカルネットワークが霧（フォグ）のように広く分散しており，その中にソフトウェアやデータ，ストレージなどのコンピューティング資源を有した IoT ゲートウェイやフォグノードを分散配置する．フォグコンピューティングとエッジコンピューティングの違いとして，フォグコンピューティングは概念的にエッジコンピューティングを含有しており，エッジコンピューティングよりもさらにエンドポイントとの距離が短い点，利用者やデバイスに近いところで処理を行うのみならず，処理に用いるコンピューティング資源を最適化すること，クラウドの技術をエッジに落とし込みながらリアルタイムに分散処理を行うことを意識している点[6]等が挙げられる．

2.1.4 メディアエッジクラウド

メディアエッジクラウド(Media-Edge Cloud, MEC)[7]は，クラウドコンピューティングシステムの発展型としてエッジコンピューティング以前に提唱された概念であり，マルチメディアアプリケーションの Quality of Service (QoS)や Quality of Experiment (QoE)を向上させることに焦点をあてている．コアネットワークから離れた，サービスの利用者に近い位置にストレージや CPU，GPU クラスタを搭載したエッジサーバを複数配置し，ユーザのコンテキスト情報およびプロファイル情報に基づいて，マルチメディアデータの並列

分散処理を行う．Contents Delivery Network (CDN)におけるエッジサーバへのデータキャッシュシステムと類似しているが，CDN では主にマルチメディアデータの送受信やキャッシュを行うのに対し，MEC ではそれらに加えて処理も行う．従来のバックボーンのデータセンタに集約されたサーバを利用するクラウド処理よりもトラフィック量を削減することができ，低遅延処理が可能となる．

MEC におけるシステム構成の代表例を以下で説明する．

1) P2P-based MEC

各エッジサーバにユーザのプロファイル，コンテキストデータおよびコンテンツデータを蓄積し，関連するユーザのデータや各コンテンツデータの位置情報は，クラスタヘッドを通して P2P でやり取りされる．図 2.5(a)のような構成をとる．

2) Central-controlled MEC

中央の Master サーバが全ユーザのプロファイル，コンテキストデータを保持し，コンテンツデータは各エッジサーバが分散して保持する．図 2.5(b)のような構成をとる．

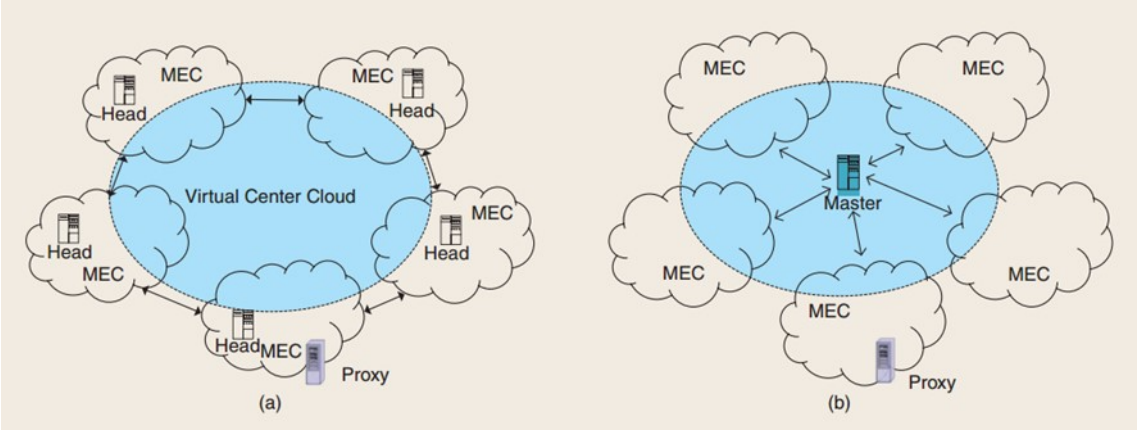

図 2.5 MEC におけるシステム構成例[7]

2.2 サーバ仮想化

サーバ仮想化とは，CPU やメモリ，ストレージといったサーバのリソースを物理的な構成にとらわれずに論理的に統合・分割することで，1 台の物理サーバ上で複数の仮想サーバを利用することができる仕組みである．物理サーバ上に仮想化のためのソフトウェアを導入することで，主に 2 種類の手法でサーバ仮想化環境を実現することができる．以下でそれぞれ説明する．

1) ホスト OS 型

仮想化ソフトウェア(仮想化アプリケーション)を用いて，物理サーバのホスト OS 上にゲスト OS を立ち上げる仮想化法方式．2)のハイパーバイザ型よりも比較的簡単に構築できるというメリットをもつ一方，ホスト OS 経由でゲスト OS 上の管理が行われるため，処理速度のオーバーヘッドが発生するといったデメリットをもつ．代表的な仮想化アプリケーションとして，Oracle VirtualBox[8]，VMware Workstation/Player/Fusion[9]，Windows Virtual PC[10]等が挙げられる．

2) ハイパーバイザ型

仮想化ソフトウェア(ハイパーバイザ)を用いて，物理サーバのハードウェア上にホスト OS を立ち上げる仮想化方式．ホスト OS を利用せずにハイパーバイザが直接ゲスト OS の管理を行うため，1)のホスト OS 型に比べて処理速度のオーバーヘッドが小さいというメリットをもつ一方，構築の難易度がホスト OS 型よりも上がるというデメリットをもつ．代表的なハイパーバイザとして，KVM[11]，Xen[12]，VMware vSphere[9]等が挙げられる．

図 2.6 仮想化サーバ手法

仮想化のメリットとして，リソースの有効活用が挙げられる．負荷のピークが異なるサーバを 1 台の物理サーバに仮想サーバという形で集約することで，物理サーバ 1 台あたりの稼働率向上と物理サーバ台数の削減が可能となり，余剰リソースや保守費用，設置スペースの削減を実現できる．また，仮想サーバの台数やスペック，稼働箇所を動的に変化させることで，システムの負荷軽減や処理能力向上，災害時のバックアップのためのスケールアウトを容易にし，移行前と同じ環境で仮想サーバを利用することができる．

一方，デメリットとしては，仮想化ソフトウェアの介在や他の仮想サーバとのリソース

共存の影響を受け，通常の物理サーバ利用時より処理性能が劣る場合がある点，仮想サーバの管理に専門的な知識が求められる点などが挙げられる．

2.3 ネットワーク仮想化

2.3.1 Software Defined Network (SDN)

SDN [13]は，ソフトウェアによってネットワークの経路選択やデータ転送を柔軟に制御する．従来の物理的なネットワーク運用では，各ネットワーク機器内にネットワークの経路制御機能とデータ転送機能が同梱されており，機器ごとに個別に設定や制御を行っていた．しかし，SDN ではネットワークの経路制御機能とデータ転送機能が分離され，SDN コントローラと呼ばれるソフトウェアによってネットワーク機器を一か所で集中的に制御することが可能となる．これにより，クラウド環境におけるネットワークの管理運用性の向上が期待できる．

SDN を実現する代表的なプロトコルに，OpenFlow[14]がある．OpenFlow では，ネットワークの経路制御機能を OpenFlow コントローラが，データ転送機能を OpenFlow スイッチがそれぞれ独立して担い，スイッチとコントローラ間の通信は OpenFlow プロトコルを用いて行われる．基本的な動作は次のようなフローで行われる．まず，OpenFlow スイッチは, OpenFlow コントローラが定めた各パケットの識別条件・処理方法等に関するルール(フローエントリ)の集合体であるフローテーブルを所持する．パケットを受信すると，パケットのヘッダ情報と各フローエントリ内のパケット識別ルールを確認する．受信したパケットのヘッダ情報がいずれかの識別条件に一致すれば，同じフローエントリ内にある処理方法に従った処理をパケットに対して行う（図 2.7).

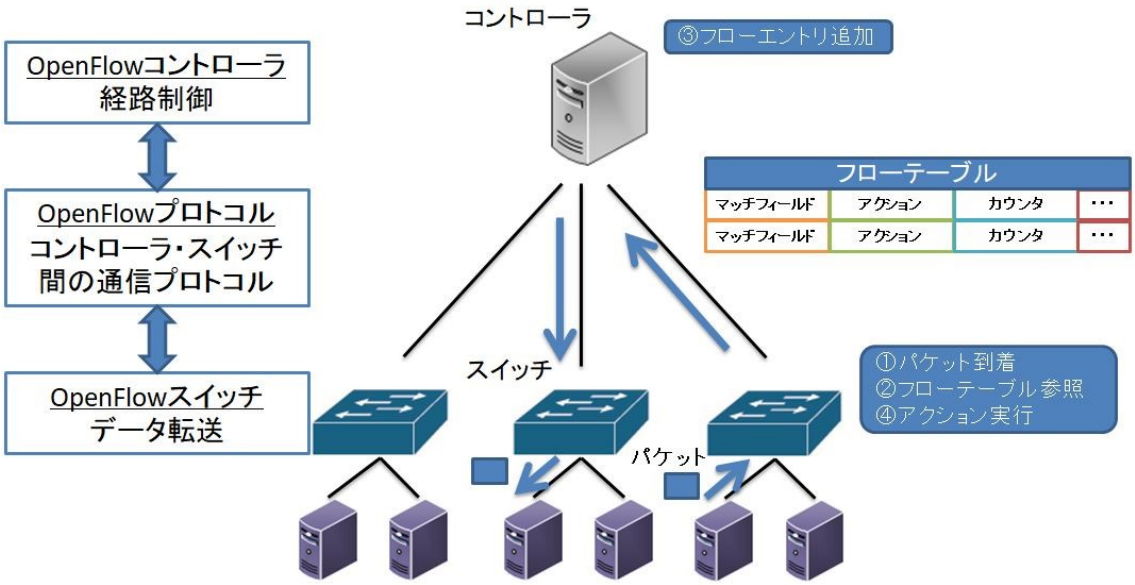

図 2.7 OpenFlow による SDN 動作例

2.3.2 Network Function Virtualization (NFV)

NFV [15]は，従来，各専用のハードウェア上で稼働していたネットワーク機能を，物理的な構成にとらわれずに，仮想化技術によって CPU・メモリ・ストレージといったハードウェアリソースが論理的に統合，分割された共用の汎用サーバ上で実現する．NFV 環境は複数のコンポーネントによって構成され，代表的なコンポーネントとして，ネットワーク機能を仮想マシン上で提供する Virtualized Network Function (VNF)，VNF 用の仮想マシンを実行するための基盤の NFV Infrastructure (NFVI)と Virtualized Network Function (VNF), VNF を起動･設定してネットワークサービス環境を自動構築する Management and Network Orchestration (MANO) (NFVI を制御する Virtualized Infrastructure Manager (VIM)，VNF の設定を行う VNF Manager (VNFM)，各自動化の中心的な役割を担う NVF Orchestrator (NFVO)を含む)が挙げられる(図 2.8).

これらのコンポーネントが組み合わさることにより，汎用サーバ上に多種多様なネットワーク機能を配置することができるため，CAPEX(設備投資費用)のコスト縮小が期待でき，合わせて運用自動化による OPEX(運用費用)のコスト縮小も期待できる．

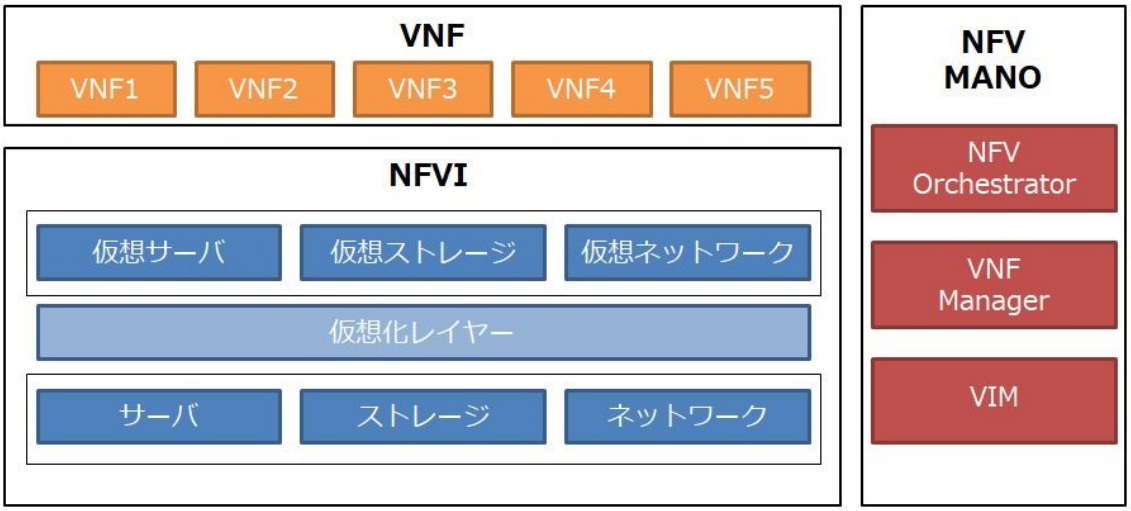

図 2.8 NFV システム概念図

2.3.3 Service Function Chaining (SFC)

SFC[16]は，NFV において，適切なサービス機能(VNF)に適切な順序でパケットを転送させるサービスチェイニングを実現するためのパケット転送方式の一つである．サービスを利用する各ユーザに対して適切なサービスを柔軟に提供するために，まず，転送するパケットにサービスを識別するためのタグをそれぞれ付与する．次に，付与されたタグに基づいてサービス機能を連結したサービスチェインを定義し，最終的にそのサービスチェインに従ってパケット転送を行う．SFC ではこれら一連の動作を IETF によって定義された 4 つの機能[17]を用いて行っており，それぞれ以下で説明する．合わせて，図 2.9 に動作手順例を示す．

1) Classifier

サービスチェインの入り口に置かれ，フローの識別，フローに適用するサービスの決定を行う．また，決定されたサービスを識別するタグをパケットに付与する．

2) Service Function Forwarder (SFF)

ネットワーク上に置かれ，タグを見てパケットを適切なサービス機能に転送する．

3) SFC Proxy

SFC のタグに対応していないサービス機能と SFF の間に置かれ，タグの取り外しと再付与を行う．

4) Controller

Classifier と SFF の管理およびテーブルの管理を行う.

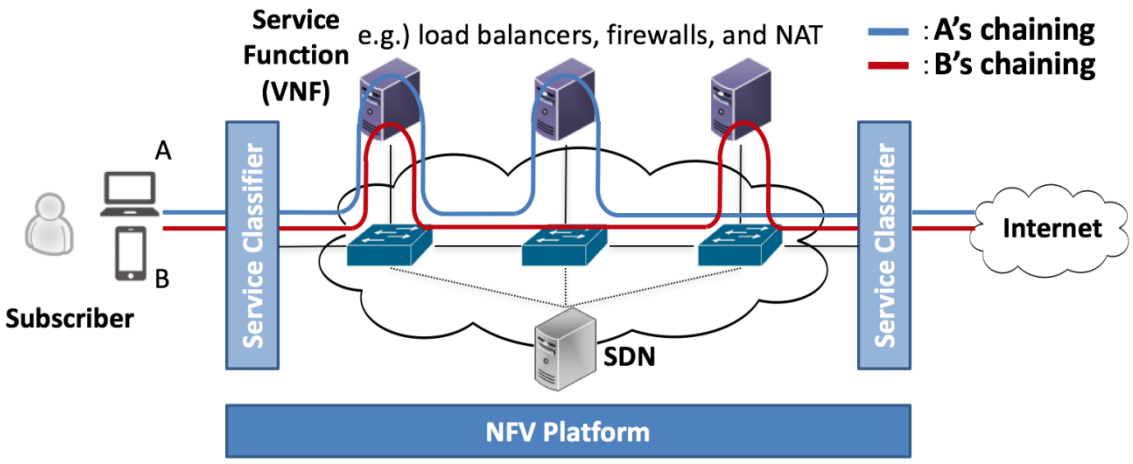

図 2.9 SFC 動作手順例

2.4 仮想クラウド環境の構築と運用

ここからは，2.1 節で紹介したコンピューティングの利用形態と 2.3 節，2.4 節で紹介したサーバ・ネットワーク仮想化が連携した仮想クラウドサービスの構築と運用について説明する.

2.4.1 パブリッククラウド

近年，クラウドプロバイダが仮想化技術をベースにクラウドインフラを作成し，商用のパブリッククラウドサービスとして提供したものをユーザが利用するケースが多くなっている[18]．クラウドの中でも特に IaaS の代表例として，Amazon が提供する Amazon Web Services[19]，Google が提供する Google Cloud Platform[20]，Microsoft が提供する Microsoft Azure[21]が挙げられる．本節では，AWS を例にとってパブリッククラウドサービスの運用に必要な代表的なサービスについて説明する.

AWS は，Amazon が社内において抱えていたビジネス課題を解決するために生まれた IT インフラストラクチャの構築実績を元に，それを企業向けに改善して 2006 年より提供を開始した Web サービスである．現在では世界 190 カ国以上，数百万の顧客を抱える大規模なサービスとなっており，強固なセキュリティと耐障害性，需要の変化に合わせたリソースの高可用性，従量課金性の料金体系，世界中に拡大するデータセンタ，膨大な提供サービ

ス数などが特徴となっている．代表的なサービスについては概要を以下で説明し，図 2.10 に各サービスの関連図を示す．また，以下で挙げるサービス以外にも，ビッグデータ分析，モバイルサービス，アプリケーションサービス，コンテンツ配信，人工知能(AI)など，企業および個人のあらゆる用途に特化したサービスを展開しており，その数は 90 以上に及ぶ(2017 年 12 月現在).

1) コンソール機能［AWS Management Console］

各サービスへのアクセスや設定のためのインターフェースをユーザに提供する．これにより，AWS アカウントに関するあらゆるクラウド管理を行うことができる.

2) コンピューティング機能［Elastic Cloud Computing (EC2)］

コンピュータ処理能力を CPU やメモリ，ストレージといった資源をもつ仮想サーバである”インスタンス”としてクラウド内で提供する．ユーザはインスタンスを起動する Region(地域)と Availability zones(データセンタの単位) (表 2.1)，OS，インスタンスタイプ(CPU 数，メモリ，ストレージがセットになったもの)を選択し，要求に合ったインスタンスを起動する．インスタンスを起動するとプライベート IP が付与され，EC2 内部のネットワーク内の通信に用いられる．オプションとしてパブリック IP アドレスを付与することも可能で，外部ネットワークからのアクセスに使用される.

3) メッセージングキュー機能［Simple Queue Service (SQS)］

AWS 上で動作するアプリケーション間でメッセージ通信を行う．これにより，アプリケーションを部品に分け，それらを独立に動作させることができる.

4) オブジェクトストレージ機能［Simple Storage Service(S3)］

大量のオブジェクトデータを保存するためのストレージサービスを提供する. REST および SOAP のインターフェースを備えている．各オブジェクトが”バケット”というコンテナの中に格納され，ユーザが指定した独自のキーを用いて読み出しや書き込み，削除，移動を行う.

5) ブロックストレージ機能［Elastic Block Storage(EBS)］

EC2 インスタンスで使用するための永続的なブロックストレージを提供する．アプリケーションからは未フォーマットの物理ディスクに見え，データベースアプリケーションやデータストレージを直接操作するようなアプリケーションに適している．また，インスタンスに関連づけられたディスクのスナップショットをとることができる.

6) データベース機能［Simple DB］

非リレーショナル型のデータストアを提供する．データの格納と照会をウェブサービス経由で行うことができる．

7) リソース計測機能［Cloud Watch］

AWS のクラウド資源と AWS 上で実行するアプリケーションの監視機能を提供する．COU 使用率や待ち時間，リクエスト数などの統計情報を収集し，ユーザの設定した閾値に応じてアラーム通知や自動アクションの実行を可能とする．

8) スケーリング機能［Auto Scaling］

必要に応じて，自動的に EC2 をはじめとした割り当て資源を変化させる．

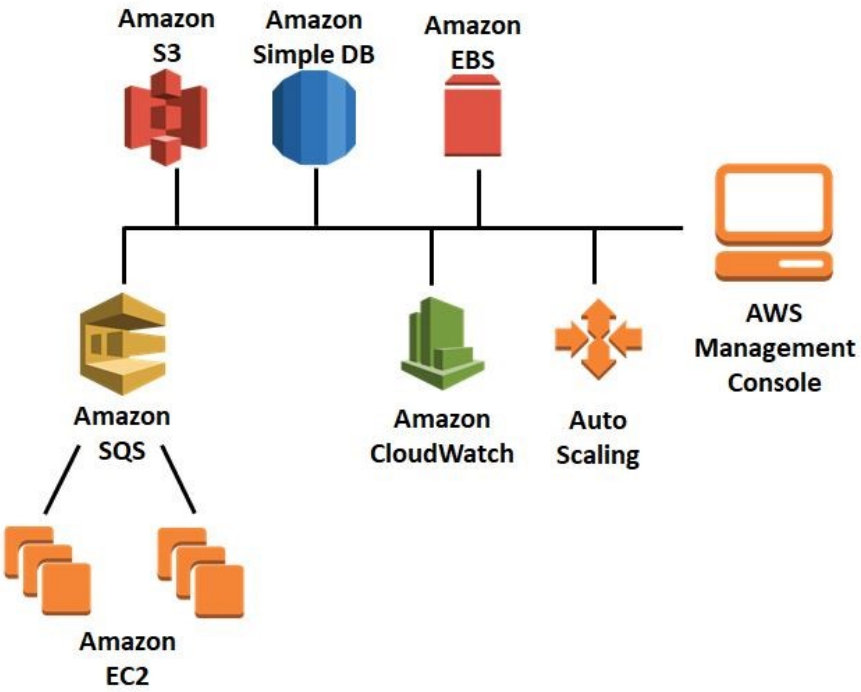

図 2.10 AWS の代表的なサービス関連図

表 2.1 AWS の Region と Availability zones (2017 年 12 月時点)[22]

Region	Location	Availability Zones
US East	North Virginia	us-east1
US East	Ohio	us-east2
US West	North California	us-west1
US West	Oregon	us-west2
Asia Pacific	Tokyo	ap-noutheast-1
Asia Pacific	Seoul	ap-nourth-2
Asia Pacific	Mumbai	ap-south-1
Asia Pacific	Singapore	ap-southeast-1
Asia Pacific	Sydney	ap-southeast-2
Canada	Central	ca-central-1
China	Beijing	cn-north-1
China	Ningxia	cn-northwest-1
EU	Frankfurt	eu-central-1
EU	Ireland	eu-west-1
EU	London	eu-west-2
EU	Paris	eu-west-3
South America	Sao Paulo	sa-east-1

2.4.2 プライベートクラウド

2.4.1 節で紹介したクラウドプロバイダが提供する商用のクラウドサービスを利用する場合，ユーザが利用できる機能はプロバイダ依存となり，プロバイダが必ずしもユーザが求める機能をサービスとして提供しているとは限らない．そういった背景から，近年，特に企業がビジネス目的にクラウドを利用する場合，自社で独自に構築したプライベートクラウドを運用するケースが増加している[23]．このプライベートクラウドを構築するためのオープンソースソフトウェアとして，OpenStack[24]，CloudStack[25]，VMware vSphere[9]などが挙げられる．本節では，OpenStack を例にとってプライベートクラウドサービスの構築と運用に必要なサービスについて説明する．

OpenStack は，米国のホスティング事業者である Rackspace Hosting 社と NASA(米国航空宇宙局)によって 2010 年に開発プロジェクトがスタートしたクラウド環境構築用のソフトウェア群であり，約半年ごとに新しいリリースが登場するサイクルで開発されている．リリース名はそれぞれアルファベット順に，そのリリースに向けて機能などを議論するカンファレンスである OpenStack Summit の開催地周辺の名称あるいは人文学上の名称とな

っており，”Austin”から現在の”Pike”までに 16 のバージョンがリリースされている(2017 年 12 月現在).

IaaS のクラウド環境として，コントローラノード，コンピュートノード，ネットワークノードやストレージノードなどの各種ノード上で仮想サーバやネットワーク，ストレージなどの各機能が実行される．それぞれの機能は容易に統合，切り離すことが可能になるよう，モジュール化されたコンポーネントとなっており，それぞれが独立して動作する分散型のシステムとなっている．そのため，システム利用者の要求にあったコンポーネントを自由に組み合わせて利用できるというメリットが挙げられる．また，OpenStack ではインスタンスに対し，リソース割り当てやマイグレーション機能を提供することができる．インスタンスはフレーバーというメモリや CPU の雛形を用いて生成され，そのフレーバーを更新するという形でインスタンスに対しリソースを追加で割り当てることができる．また，マイグレーションとは，あるコンピュートノードから別のコンピュートノードへインスタンスを移行する機能である．OpenStack 上でこれらの機能を活用することで，状況に応じてインスタンスのスケーリングを図ることができる．

以下でそれぞれのコンポーネントの概要[26]を説明し，図 2.15 にコンポーネントの全体像を示す.

1) コンソール機能 [OpenStack Dashboard, Horizon]

Web ブラウザから利用する，GUI のダッシュボードを提供する．実際の環境操作は，Horizon から各種コンポーネントの API にリクエストを送信することで行われる．基本的にはクライアントツールにおる CUI 操作の方が作業効率は高くなるが，仮想ネットワークの構成やインスタンス(仮想マシン)のコンソール表示など，Horizon を使った方が便利な部分もある．

2) コンピューティング機能 [OpenStack Compute, Nova]

ユーザからのリクエストに応じて，コンピュートノードでインスタンスを起動する．ユーザは事前に定義されたインスタンスタイプ(フレーバー)からインスタンスのサイズを選択する．合わせて，インスタンスに適用するセキュリティグループ，SSH ログイン用の認証鍵, 接続先の仮想スイッチ, 起動するゲスト OS のテンプレートイメージなどを指定する.

3) イメージ機能 [OpenStack Image Service, Glance]

ゲスト OS が導入されたテンプレートイメージのカタログを管理する．ユーザは，管理者が事前に登録したイメージを利用するだけでなく，自分で作成したイメージを Glance の API を介してアップロードすることもできる．また，起動中のインスタンスについて，OS

領域のディスク(ルートディスク)を複製して，新たなイメージとして登録することも可能である(スナップショット機能，図 2.11).

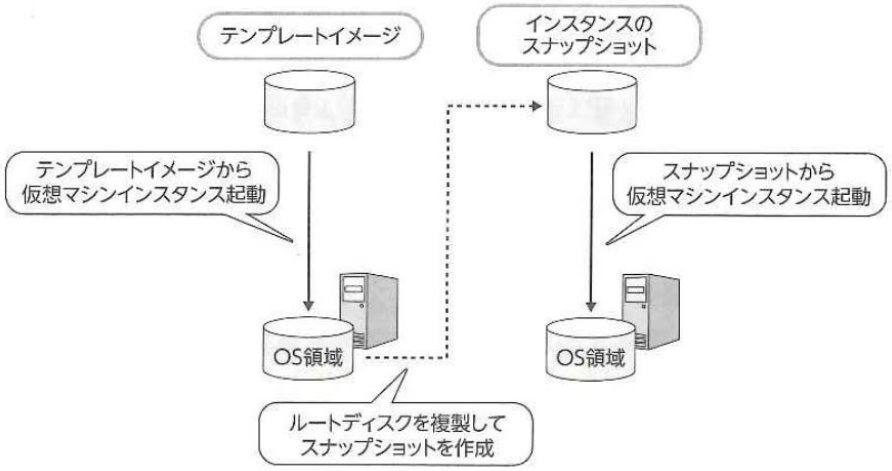

図 2.11 Glance におけるインスタンスのスナップショット[26]

4) 認証機能［OpenStack Identity, Keystone］

OpenStack 全体の認証機構を提供する．ユーザは各コンポーネントの API にリクエストを送信する際，事前に Keystone の API で認証を行う．この際に発行されるトークン(API の利用許可証)の ID 番号と合わせて, 目的のコンポーネントに API リクエストを送信する．

5) ネットワーク機能［OpenStack Networking, Neutron］

プロジェクトごとに独立した仮想ネットワークを提供する．ユーザは Neutron の API を介して仮想ルーターや仮想 L2 スイッチなど，仮想的なネットワークコンポーネントを追加していく．典型的な構成例としては，外部ネットワークに対してプロジェクト専用の仮想ルーターを接続しておき，その配下に仮想 L2 スイッチを接続していく．

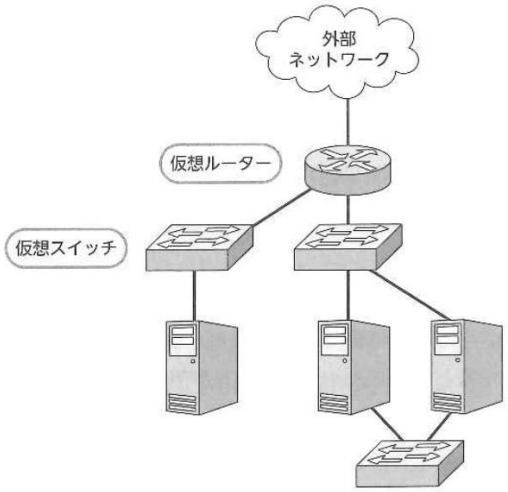

図 2.12 Neutron における仮想ネットワークの構成例[26]

6) オブジェクトストレージ機能［OpenStack Object Storage, Swift］

ファイル単位でデータを出し入れするオブジェクトストレージサービスを提供する．インスタンスに接続するわけではなく，Swift の API を介してファイルをアップロード/ダウンロードするという使い方をする．耐障害性と拡張性を重視したアーキテクチャとなっており，ファイルの実体を複数のサーバに分散保存することで，多数のクライアントからの複数のファイルに対するアクセス時も，システム全体として高いスループットを実現する．

7) ブロックストレージ機能［OpenStack Block Storage, Cinder］

永続データを保存するためのブロックボリュームストレージサービスを提供する．ユーザは Cinder の API を介して事前に任意のサイズのブロックボリュームを作成しておき，作成したブロックボリュームを起動中のインスタンスに接続すると，ゲスト OS からは追加のディスク領域として認識される．また，既存のブロックボリュームのスナップショットコピーを作成しておき，そこから同じ内容のブロックボリュームを複製することも可能である．さらに，別の利用方法として，ブロックボリュームを用いたインスタンスの起動も可能である．この場合，Glance が管理するテンプレートイメージの内容をコピーしたブロックボリュームを事前に作成しておき，これをインスタンスに接続してブロックボリューム内のゲスト OS を起動する．インスタンスを破棄した後でも OS 領域のディスクを残しておきたい場合に有効である．

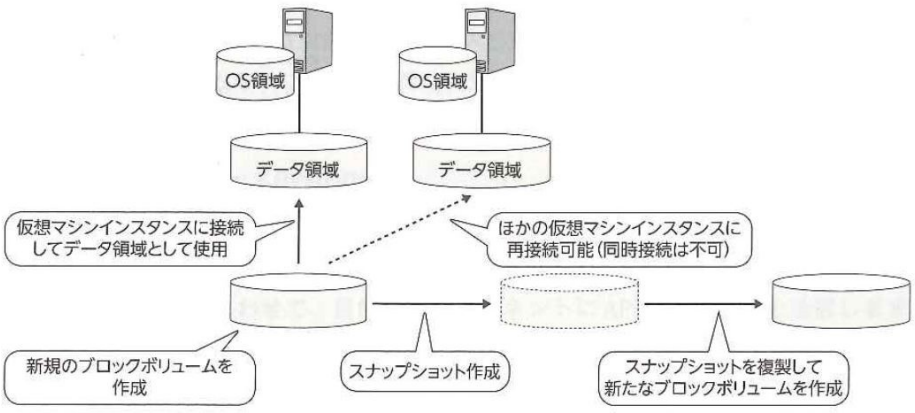

図 2.13 Cinder におけるブロックボリュームの利用方法[26]

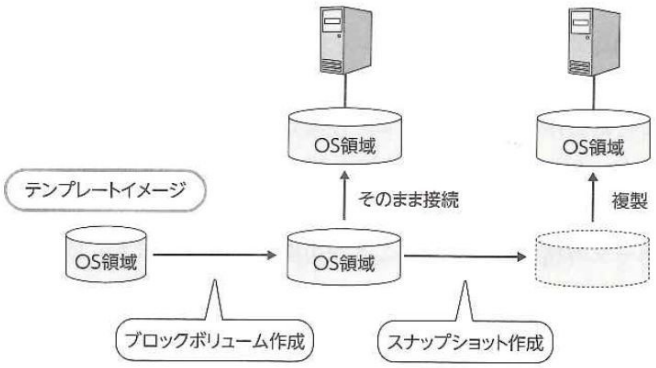

図 2.14 Cinder におけるブロックボリュームからのインスタンス起動[26]

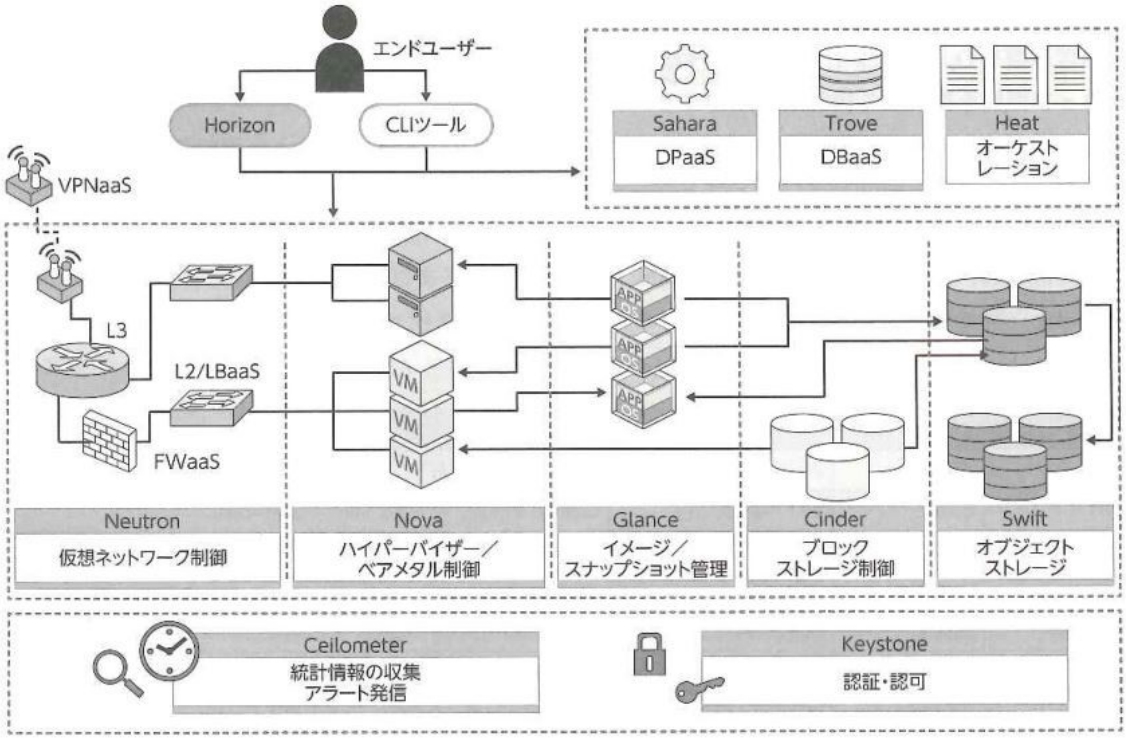

図 2.15 OpenStack を構成するコンポーネントの全体像[26]

第 3 章 遅延分析モデルを用いたエッジコンピューティングにおけるマルチメディア処理実行遅延評価

本章では，従来のクラウドコンピューティングおよびエッジコンピューティング環境においてマルチメディア処理を実行する際の遅延特性をシナリオ別に理論モデル化し，それぞれの遅延分析を行う．さらに，エッジコンピューティング環境において，ネットワークの使用状況や計算資源を踏まえてさらなる低遅延処理を実現するためのネットワーク経路の最適化を試みる．OpenFlow コントローラを利用することで，最適化されたネットワーク経路の制御を行い，クラウドコンピューティングによる処理と比較し，エッジコンピューティングの有効性を示す．

3.1 遅延分析モデル

本節では，図 3.1 のようなコンピューティング環境において，従来のクラウドコンピューティングで使用するコアサーバでの処理とエッジサーバによる分散処理の遅延分析モデルを検討する．モデル式で用いるパラメータを，以下表 3.1 に定義する．

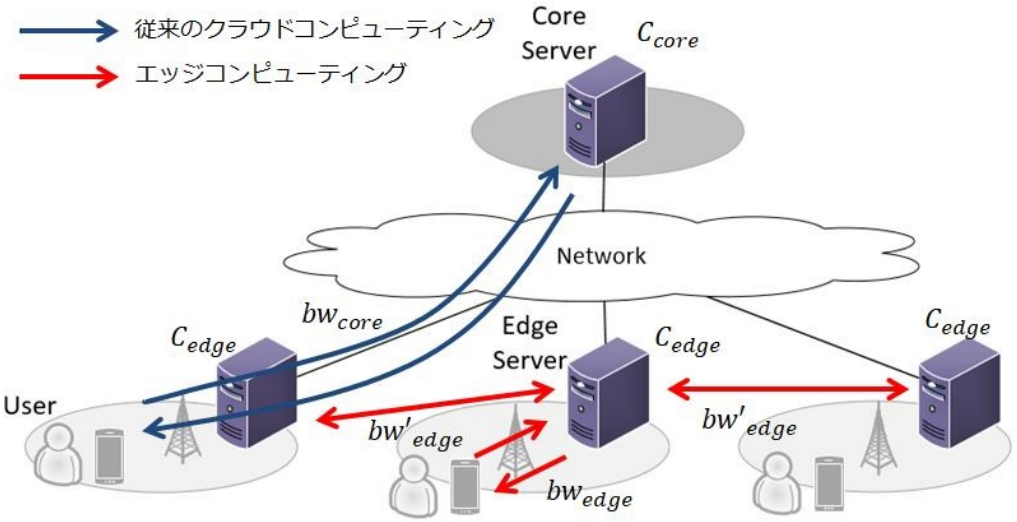

図 3.1 従来のクラウドコンピューティングおよびエッジコンピューティング環境

本章は文献[K. Imagane et al., “Evaluation and Analysis of System Latency of Edge Computing for Multimedia Data Processing,” IEEE GCCE, Oct.2016.], [今金 等., “エッジコンピューティングを活用したマルチメディア分散処理のための遅延評価,” IEICE CS 研究会, 2016 年 12 月.]に基づいている．

表 3.1 遅延分析モデルのパラメータ定義

パラメータ［単位］	定義
L_{total} [s]	総遅延時間
L_{proc} [s]	アプリケーション処理遅延
L_{comm} [s]	データ通信遅延
n [台]	エッジサーバ数
m [回]	処理サイクル数
$L_{core,edge}$ [s]	コア・エッジでの実行遅延
D_{tx} [Mbit]	送信データサイズ
D_{rx} [Mbit]	受信データサイズ
D_i [Mbit]	各エッジサーバ間の送信データサイズ
$bw_{core,edge}$ [Mbps]	コア/エッジ NW までのスループット
bw'_{edge} [Mbps]	エッジ NW 内のスループット
$C_{core,edge}$ [clock cycle/s]	コア・エッジサーバの処理速度
O [clock cycle]	処理プロセス数

まず，単純化のため，ユーザ端末 1 台，エッジサーバ 1 台，コアサーバ 1 台の環境を想定し(図 3.2，これをエッジ 1 台シナリオとする．総遅延時間[s](*L*$_{total}$)は，コア・エッジサーバでのアプリケーション処理遅延[s](*L*$_{proc}$)と，ユーザとコア・エッジサーバ間のデータ通信遅延[s](*L*$_{comm}$)の和として表現できる．

$$L_{total} = L_{proc} + L_{comm} \tag{3.1}$$

ここで，アプリケーション処理遅延[s](*L*$_{proc}$)は，アプリケーション処理に必要な処理プロセス数[clock cycle](*O*)と各サーバの処理速度[clock cycle/s](*C*)に依存し，次のように表現できる．

$$L_{proc} = \frac{O}{C} \tag{3.2}$$

同様に，通信遅延[s]（*L*$_{comm}$)は，処理のために各サーバが受信するデータ[Mbit](*D*$_{rx}$)と，各サーバが送信する処理結果データ[Mbit]（*D*$_{tx}$)，ユーザと各サーバ間のスループット[Mbps](*bw*)に依存し，次のように表現できる．

$$L_{comm} = \frac{D_{rx} + D_{tx}}{bw} \tag{3.3}$$

以上より，従来のクラウドコンピューティングにおける実行遅延[s] (L_{core})，エッジコンピューティングにおける実行遅延[s] (L_{edge})は，式(3.2)，(3.3)を式(3.1)に代入してそれぞれ次のように表現できる．

$$L_{core} = \frac{O}{C_{core}} + \frac{D_{rx} + D_{tx}}{bw_{core}} \tag{3.4}$$

$$L_{edge} = \frac{O}{C_{edge}} + \frac{D_{rx} + D_{tx}}{bw_{edge}} \tag{3.5}$$

なお，C_{core} ，bw_{core} はそれぞれコアサーバの処理速度[clock cycle/sec]，ユーザとコアサーバ間のスループット[Mbps]とし，同様に，C_{edge} ，bw_{edge} はそれぞれエッジサーバの処理速度[clock cycle/sec]，ユーザとエッジサーバ間のスループット[Mbps]とする．また，一般的に $C_{edge} > C_{edge}$，$bw_{edge} < bw_{edge}$ が成立する．今回のモデルでは，各サーバは，ローカル端末よりD_{rx}のサイズを持つデータを受信し，計算量Oの処理を行い，ローカル端末へD_{tx}のデータを返信するという想定である．また，ローカル端末では，連続して処理したいデータが生成されることを想定し，一連の流れを m 回繰り返すこととする．以上を踏まえると，(3.4)，(3.5)式より，クラウドコンピューティングとエッジコンピューティングにおける実行遅延差[s]は式(3.6)のように処理遅延と通信遅延の差の和で表現でき，ユーザとコアサーバ間のスループット(bw_{core})がユーザとエッジサーバ間のスループット(bw_{edge})に比べて小さくなるほど，また，コア・エッジサーバの処理性能差が開くほどエッジコンピューティングによる処理が優位となることがわかる．

$$\begin{aligned} L_{core} - L_{edge} = O \times m \times \left(\frac{1}{C_{core}} - \frac{1}{C_{edge}}\right) \\ +(D_{rx} + D_{tx})\left(\frac{1}{bw_{core}} - \frac{1}{bw_{edge}}\right) \end{aligned} \tag{3.6}$$

図 3.2 ユーザ端末 1 台，エッジサーバ 1 台，コアサーバ 1 台の環境
(シングルエッジシナリオ)

次に，ユーザ端末 1 台，エッジサーバ複数台，コアサーバ 1 台の環境を想定し(図 3.3, これをパイプラインシナリオと定義する．パイプラインシナリオでは，ローカル端末からデータを 1 台のエッジサーバに転送し，そこで一次的な処理O_1を施した後に次のエッジサーバにデータD_1を転送して別の二次的な処理を行う．この処理を全処理が完了するまで繰り返し，最後の処理を行ったエッジサーバがローカル端末に処理データを転送する．実行遅延L_Eは式(3.7)のように表すことができ，ローカル端末のデータ送受信時間と各エッジサーバでの処理時間，エッジサーバ間のデータ転送時間に，各処理時間とデータ送信時間の最大値の蓄積分を加えたものとなる．なお，各サーバの性能，サーバ間帯域は等しいと仮定する．

$$L_{edge} = \frac{D_{rx}}{bw_{edge}} + max\left(\frac{D_{rx}}{bw_{edge}}, \frac{O_1}{C_{edge}}, \dots, \frac{O_n}{C_{edge}}\right) \times (m-1) + \frac{\sum_{i=1}^{n} O_i}{C_{edge}} + \frac{\sum_{i=1}^{n-1} D_i}{bw'_{edge}} + \frac{D_{tx}}{bw_{edge}} \tag{3.7}$$

また，パイプラインシナリオにおけるコアサーバ 1 台，エッジサーバ複数台(n 台)での実行遅延差は式(3.8)のように表すことができる．

$$L_{core} - L_{edge} = (D_{rx} + D_{tx})\left(\frac{1}{bw_{core}} - \frac{1}{bw_{edge}}\right) + O \times \left(\frac{m}{C_{core}} - \frac{1}{C_{edge}}\right) - \frac{\sum_{i=1}^{n-1} D_i}{bw'_{edge}} - max\left(\frac{D_{rx}}{bw_{edge}}, \frac{O_1}{C_{edge}}, \dots, \frac{O_n}{C_{edge}}\right) \times (m-1) \tag{3.8}$$

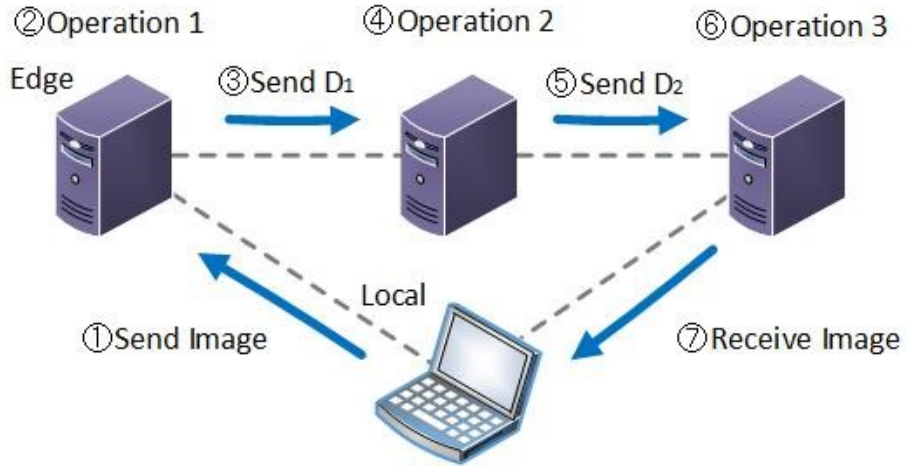

図 3.3 ユーザ端末 1 台，エッジサーバ複数台，コアサーバ 1 台の環境
(パイプラインシナリオ)

次に，ユーザ端末 1 台，エッジサーバ複数台，コアサーバ 1 台の環境を想定した(図 3.4 並列処理シナリオを定義する．並列処理シナリオでは，ローカル端末からデータを全エッジサーバへ順に転送し，それぞれのエッジサーバで全処理を行うものとする．各サーバはシングルエッジシナリオと同じ処理を行い，台数が増えるほど処理時間が短くなると考えられる．なお，パイプラインシナリオ同様，各サーバの性能は等しいと仮定する．

$$L_{edge} = \frac{D_{rx} \times m}{bw_{edge}} + \frac{\sum_{i=1}^{n} O_i \times m}{C_{edge} \times n} + \frac{D_{tx}}{bw_{edge}} \tag{3.9}$$

また，パイプラインシナリオにおけるコアサーバ 1 台，エッジサーバ複数台(n 台)での実行遅延差は式(3.10)のように表すことができる．

$$L_{core} - L_{edge} = (D_{rx} \times m + D_{tx}) \left(\frac{1}{bw_{core}} - \frac{1}{bw_{edge}} \right) + O \times m \times \left(\frac{1}{C_{core}} - \frac{1}{n \times C_{edge}} \right) \tag{3.10}$$

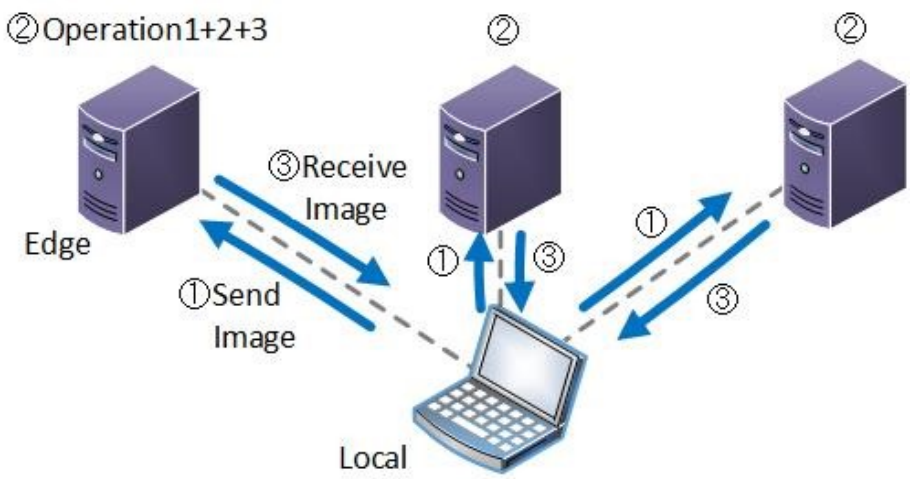

図 3.4 ユーザ端末 1 台，エッジサーバ複数台，コアサーバ 1 台の環境
(並列処理シナリオ)

3.2 遅延評価

本節では，従来のクラウドコンピューティングおよびエッジコンピューティング環境でマルチメディア処理を実行する際の遅延特性について, 3.1 節の遅延分析モデルおよび 3.2.1 節のマルチメディアアプリケーションを用いて，特性評価および実機評価を行う．従来のクラウドコンピューティング環境として，IaaS のクラウドプロバイダのサービスである Amazon の「Amazon EC2[29]」を，エッジコンピューティング環境として，さくらインターネットの「さくらクラウド[30]」を利用する．また, 研究室内の PC をユーザ端末とする．ユーザ端末から画像アプリケーション処理を上記のエッジサーバ，コアサーバにオフロードし，処理結果を返すまでの時間に関して，3 つのシナリオ(シングルエッジ，パイプライン，並列分散)での比較評価を行う．

表 3.2 エッジサーバ，コアサーバの概要

サーバ	サービス	CPU	メモリ	OS
コアサーバ	Amazon EC2	Intel® Xeon® CPU E5-2670 v2 @2.50GHz	4GB	Ubuntu14.04
エッジサーバ	さくらクラウド	Intel® Xeon® E5-2650@2.30GHz	1GB	Ubuntu14.04

3.2.1 マルチメディアアプリケーション

従来のクラウドコンピューティングおよびエッジコンピューティング環境において実行するアプリケーションとして，マルチメディア処理である人物検出アプリケーションを実装した．本節では，後述の実験の導入として，そのアプリケーションの概要について説明する．

1) イメージモザイキング

従来のクラウドコンピューティングおよびエッジコンピューティング環境において実行するアプリケーションとして, 図 3.5 に示すような OpenCV[27]を利用したイメージモザイキングを適用する．

イメージモザイキングは，複数の画像を合成して，視野が広く高解像度のパノラマ画像を生成する技術である．一般的なイメージモザイキングの手法の一つである，特徴点のマッチングを用いた手法では，図 3.6 のようなフローで処理を行う．一方，OpenCV (Open Source Computer Vision Library)は，インテルが開発したオープンソースのコンピュータビジョン向けライブラリで，C/C++，Java，Python に対応している．プラットフォームとして Mac OS，Windows，Linux，iOS，Android などをサポートしており，画像処理や構造解析，機械学習などが利用できる．

本研究では，エッジサーバとコアサーバに OpenCV 環境を構築し，イメージモザイキングのサンプルプログラムとして提供されている stitching.cpp に対して，各サーバでの処理時間を計測，表示するプログラムを書き加えて実装する．

図 3.5 イメージモザイキングの実行例

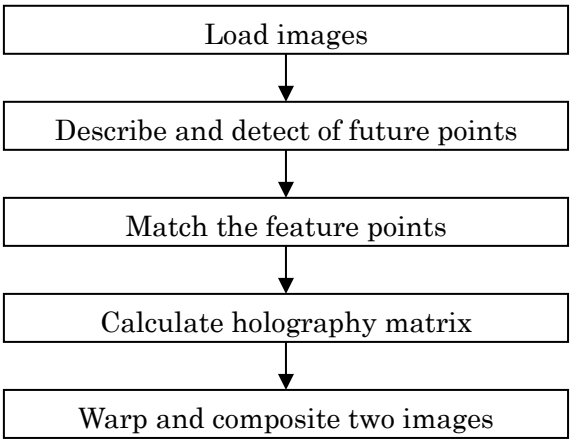

図 3.6 特徴点ベースのイメージモザイキングフロー

2) 赤外線熱画像を用いた可視光画像の人物検出

イメージモザイキング同様，従来のクラウドコンピューティングおよびエッジコンピューティング環境において実行するアプリケーションとして，赤外線熱画像を用いた人物検出処理を適用する．画像からの人物検出は，従来，特徴量抽出や機械学習手法の組み合わせにより行う．代表例として，画素の輝度勾配情報を利用する HOG 特徴量による人物検出法が挙げられるが，検出精度が高いとはいえず，誤検出や検出漏れが避けられないのが現状である．そこで，本手法では HOG 特徴量による人物検出での誤検出分を，同時に撮影した赤外線熱画像を用いて除去することで人物検出精度の改善を目指す．

本研究では，1)で構築した OpenCV 環境の下，図 3.7 のフローで処理を行うプログラムを実装する．まず，ある人物を可視光カメラと赤外線カメラで撮影し，サイズ調整を行った可視光画像と赤外線熱画像の位置合わせを透視投影変換で行う．次に，HOG 特徴量を用いて可視光画像の人物検出を行い，この検出結果に対して温度情報による判定を行う．なお，判定は，人物が検出された箇所において矩形内の画素の 25%以上 60%以下が人間の体温に近い温度を示すかどうかを基準とする．

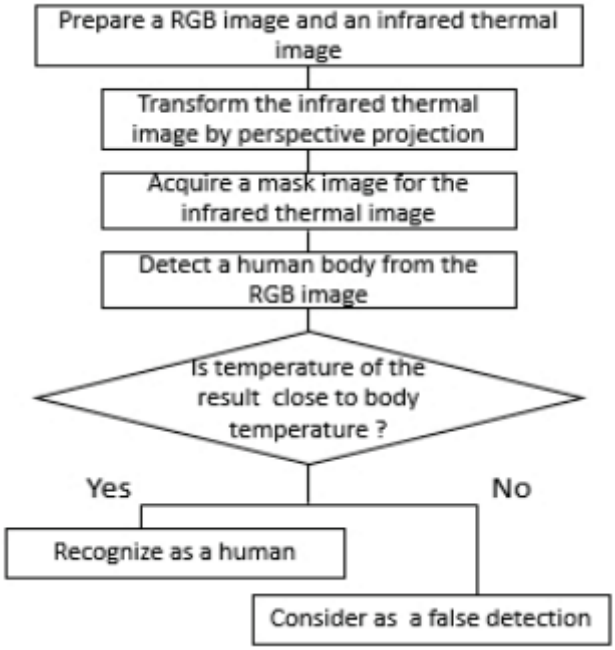

図 3.7 赤外線熱画像を用いた人物検出処理フロー[28]

(a)HOG 特徴量による検出　(b)可視光・赤外線画像合成　(c)判定結果

図 3.8 赤外線熱画像を用いた可視光人物検出実行例

3.2.2 遅延分析モデルを用いた特性評価

本節では，遅延分析モデルに対して複数のパラメータを付与することで，従来のクラウドコンピューティングおよびエッジコンピューティング環境のマルチメディア処理実行遅延特性を理論的に評価する．ここでは，3.2.1 節 1)で紹介したイメージモザイキングの実行を想定する．イメージモザイキングの手順としては，まずイメージモザイキングに適用する画像として静止画を GoPro で撮影し，撮影画像を端末からコアサーバ，エッジサーバへ転送する．その後，それらの画像を用いてサーバ上でイメージモザイキング処理を行い，端末へ生成画像を転送する．

4K 画像 6 枚より 1 枚のパノラマ画像を生成するイメージモザイキングを行うと仮定して，従来のクラウドコンピューティングおよびエッジコンピューティング環境のマルチメディア処理実行遅延差を表す式(3.6)，(3.8)，(3.10)に複数条件からなるパラメータを代入していく(D_{tx}=216[Mbit]，D_{rx}=24[Mbit]，O =40[clock cycle]，その他のパラメータは条件によって可変)．なお，ここでは分散された各計算量$O_1, O_2,,, O_n$は等しいものとなっている．処理回数 m を変化させたときの実行遅延－処理速度ゲイン特性を図 3.9~3.11 に示す．X 軸は各サーバの処理速度のゲイン(C_{edge}/C_{core})を，Y 軸は実行遅延のゲイン($L_{core}-L_{edge}$)を表している．

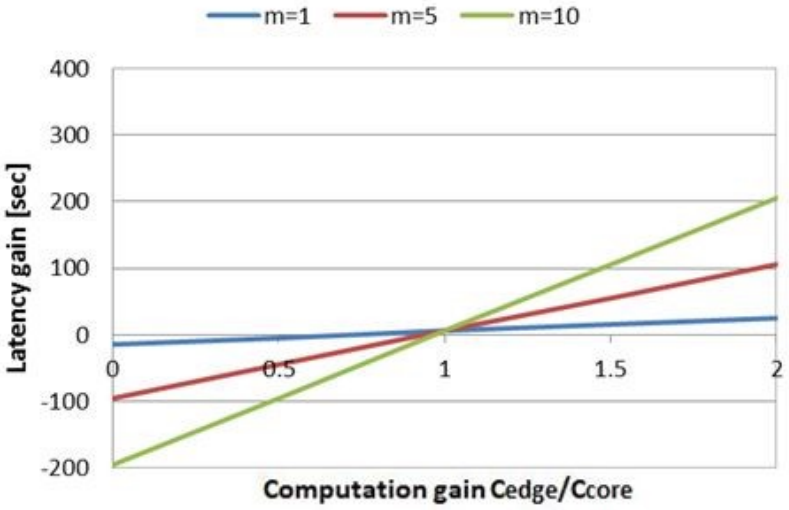

図 3.9 処理回数を変化させたときの実行遅延－処理速度ゲイン特性
シングルエッジシナリオ（n=1）

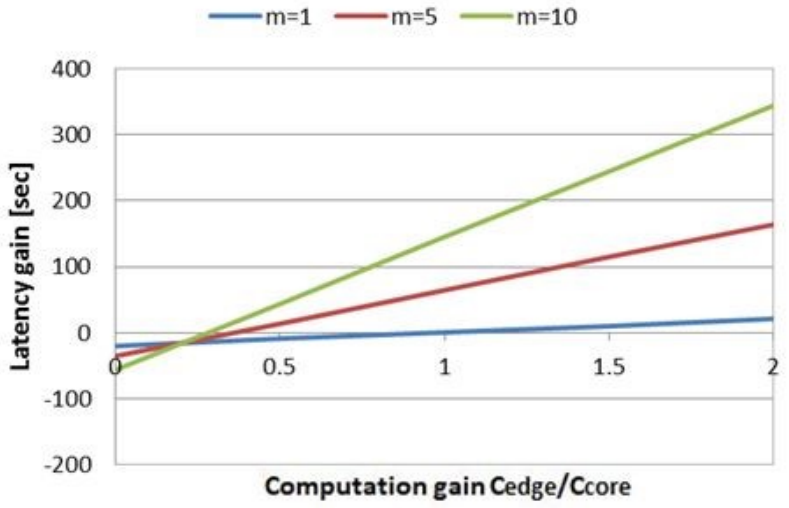

図 3.10 処理回数を変化させたときの実行遅延－処理速度ゲイン特性
パイプラインシナリオ（n=5）

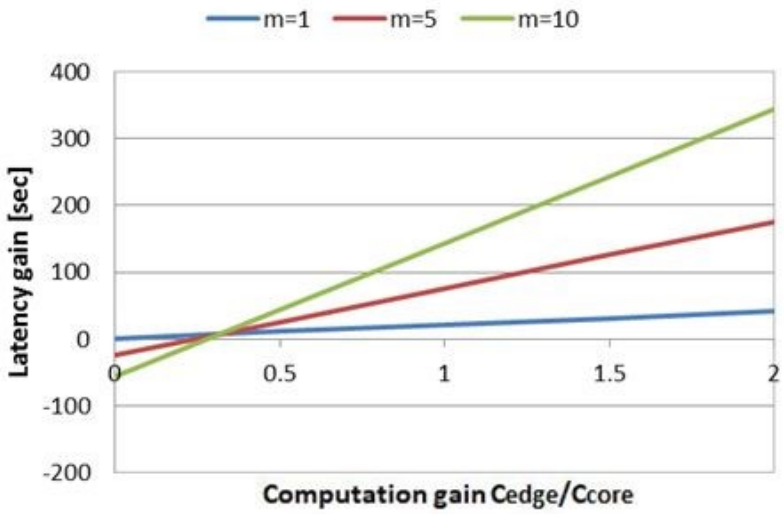

図 3.11 処理回数を変化させたときの実行遅延－処理速度ゲイン特性
並列処理シナリオ（n=5）

図 3.9~3.11 を比較すると，本アプリケーションにおいては，いずれのシナリオにおいても，各サーバの処理性能差が開き，処理回数が多くなるほどエッジサーバでの処理時間が優位となることがわかる．エッジサーバ 1 台での処理の際は各処理回数の交点が横軸 1 付近に位置しているのに対し，エッジサーバ 5 台での処理の際は交点が横軸 0.2~0.3 付近に推移している．また，以下図 3.12，3.13 ではパイプライン，並列処理シナリオで n=10 の場合の実行遅延－処理速度ゲイン特性を示しており，エッジサーバが 10 台での処理の際は横軸 0.1~0.2 付近に推移していることがわかる．したがって，エッジサーバを増やしていくと交点は横軸 1/n 付近に推移し，処理能力が低いエッジサーバでも，複数台利用することによって短時間での処理が可能になることが確認できる．

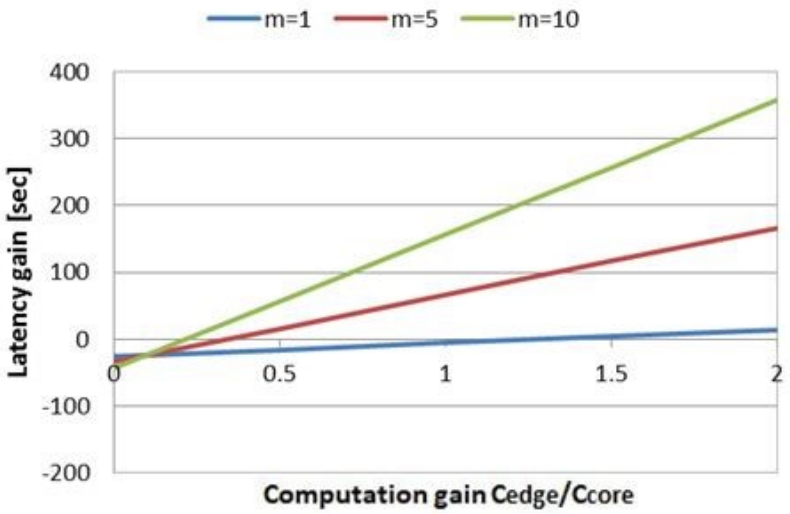

図 3.12 処理回数を変化させたときの実行遅延－処理速度ゲイン特性
パイプラインシナリオ（n=10）

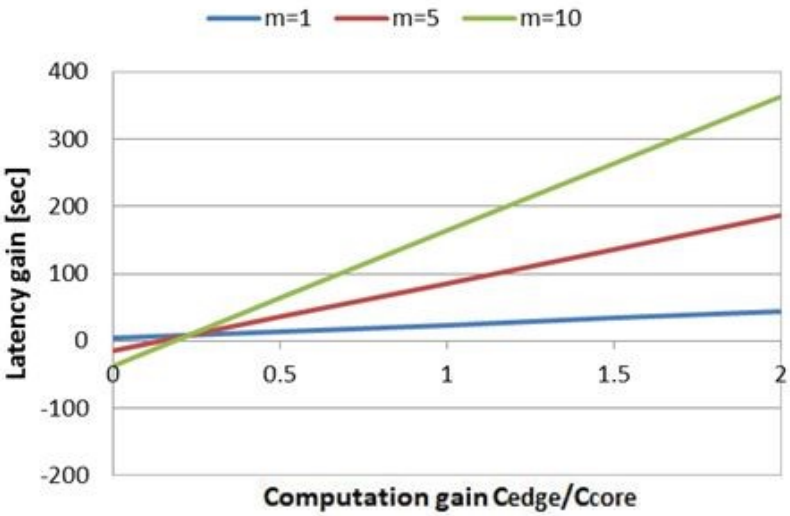

図 3.13 処理回数を変化させたときの実行遅延－処理速度ゲイン特性
並列処理シナリオ（n=10）

次に，帯域($bw_{core,edge}$)を変化させたときの実行遅延－処理速度ゲイン特性を図 3.14~3.16 に示す.

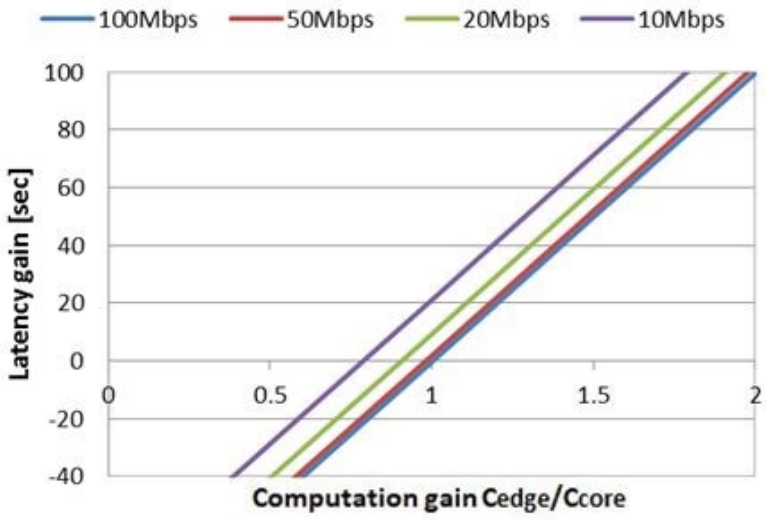

図 3.14 端末－コアサーバ間の帯域を変化させたときの実行遅延－処理速度ゲイン特性
シングルエッジシナリオ（n=1, m=5）

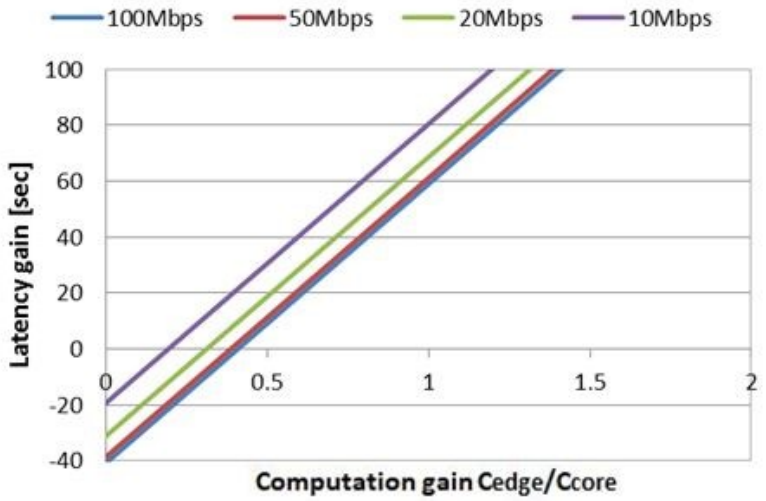

図 3.15 端末－コアサーバ間の帯域を変化させたときの実行遅延－処理速度ゲイン特性
パイプラインシナリオ（n=5, m=5）

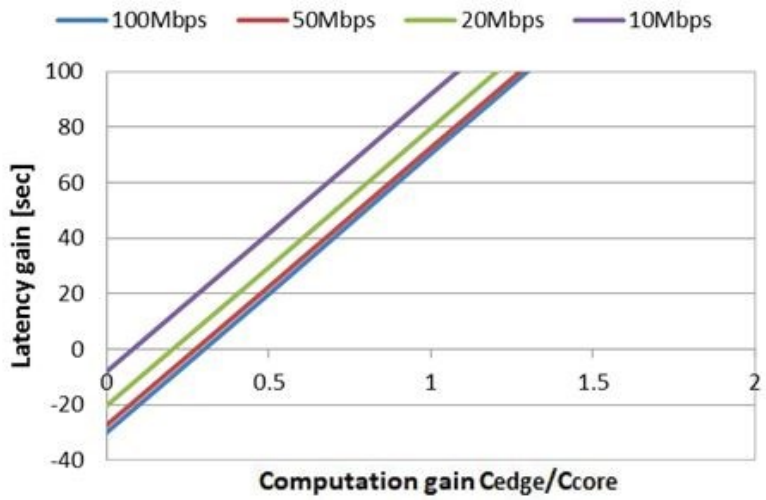

図 3.16 端末－コアサーバ間の帯域を変化させたときの実行遅延－処理速度ゲイン特性
並列処理シナリオ（n=5, m=5）

理論式から推測できるとおり，いずれのシナリオにおいても，ローカル端末－コアサーバ間の帯域が狭まるほどエッジサーバでの処理時間が優位となることがわかる．パイプラインシナリオでは，エッジサーバの台数を増やすとエッジサーバ間の通信が発生するため，通信時間において分散処理は不利になるが，その分全体の処理時間が通信時間の増加量を十分相殺できるほど短くなる．また，実クラウドサービスを用いる場合はエッジサーバが同データセンタ内にあることが多く，エッジサーバ間の帯域bw'_Eの値はローカル端末－エッジサーバ間の帯域bw_Eに比べて大きい，つまり，エッジサーバ間はより広い帯域で通信できるようになるため，パイプラインシナリオにおける全体の実行遅延はより短くなると考えられる．

本節では，3.1 節で検討した遅延分析モデルに対して，ユーザ端末よりイメージモザイキング処理のオフロードを行ったと仮定して，理論式のパラメータに数値を代入し，アプリケーション実行遅延を評価した．処理能力が低いエッジサーバでも，複数台で分散処理を実施することによってアプリケーション実行遅延の短縮化が可能になり，ユーザとコアサーバ及びエッジサーバ間の帯域差や両サーバの処理性能差が開くほど短縮化のメリットが現れるといった遅延特性を理論的に確認できた．

3.2.3 実機評価 1

本節では，エッジコンピューティング環境でイメージモザイキング処理を行うことを想定し，生成するパノラマ画像の枚数を変化させたときの遅延特性を遅延分析モデルで評価し，実機評価の結果と比較する．なお，イメージモザイキングは，エッジサーバ上で分散される計算量が等しくなるといった特徴を持つアプリケーションとなっている．

パノラマ画像 1 枚に対して入力画像は 4K 画像 6 枚とする．生成するパノラマ画像を 5,10,15 枚とし，エッジ 1 台での処理実行時間を図 3.17 に，エッジサーバ数 5,10 台としたときのパイプラインシナリオと並列処理シナリオの処理実行遅延を図 3.18，3.19 にそれぞれ示す．

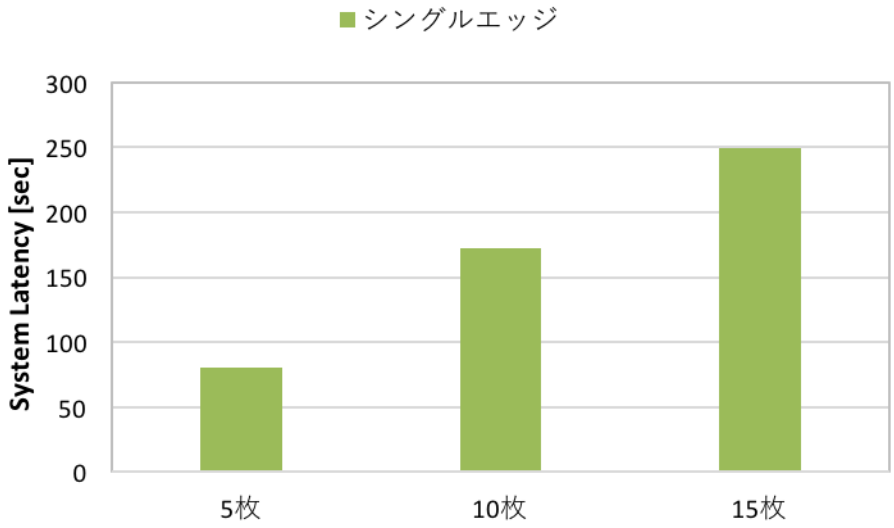

図 3.17 シングルエッジシナリオでの処理実行遅延特性 (n=1)(理論値)

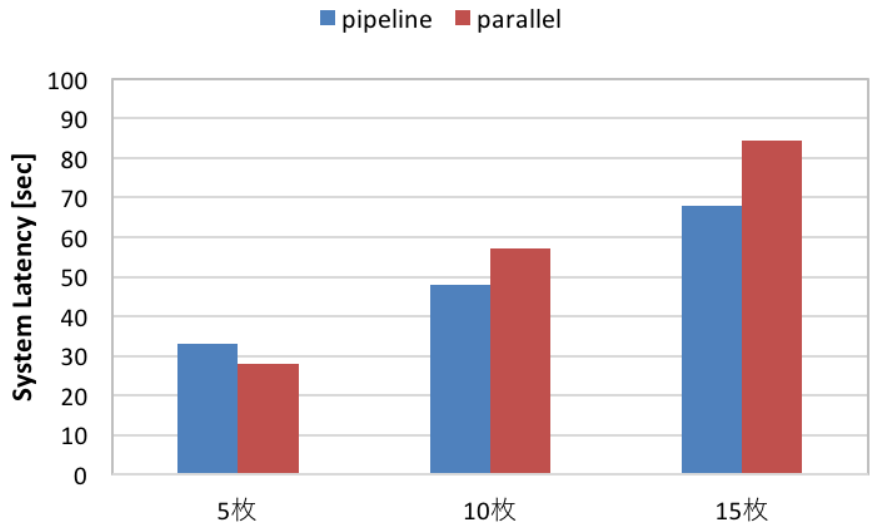

図 3.18 パイプライン・並列処理シナリオでの分散処理実行遅延比較（n=5)(理論値)

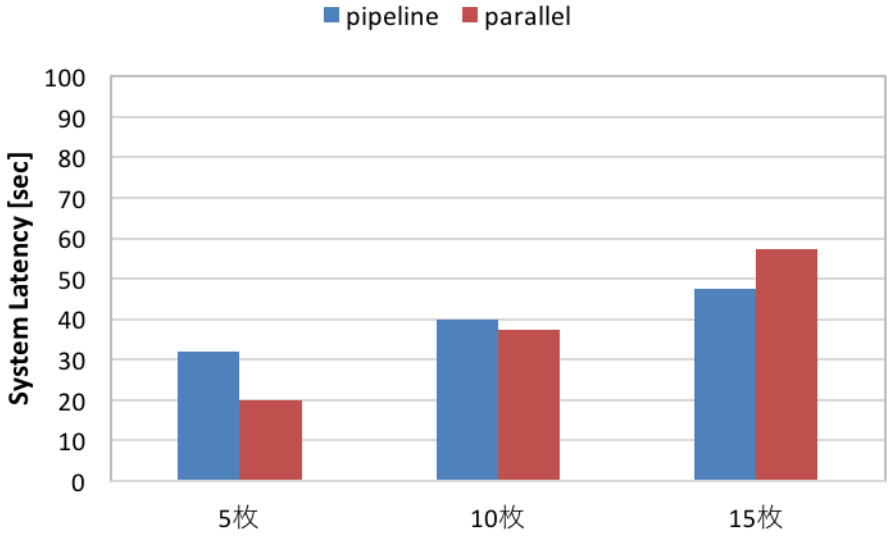

図 3.19 パイプライン・並列処理シナリオでの分散処理実行遅延比較（n=10)(理論値)

図 3.17 のシングルエッジシナリオでの実行時間は生成画像 5 枚で 80.0 秒，10 枚で 172.3 秒, 15 枚で 249.7 秒となっており, 図 3.18, 3.19 ではいずれもこれらの値を下回っている．さらに，エッジ 10 台での処理にかかる時間はエッジ 5 台での処理にかかる時間をいずれも下回っている．以上の 2 点より，両シナリオともにサーバ台数を増やすことで実行時間を大幅に短縮できることがわかる．一方，パイプラインシナリオの実行時間から並列処理シナリオの時間を引くと式(3.11)のようになる．

$$\frac{\sum_{i=1}^{n-1} D_i}{bw'_{edge}} + \left\{ max\left(\frac{D_{rx}}{bw_{edge}}, \frac{O_1}{C_{edge}}, \ldots, \frac{O_n}{C_{edge}} \right) - \frac{D_{rx}}{bw_{edge}} \right\} \times (m-1) + \frac{O \times \left(1 - \frac{m}{n}\right)}{C_{edge}} \quad (3.11)$$

生成画像の枚数が少ない 5 枚では，共に実行時間はパイプラインシナリオ > 分散処理シナリオという関係であるのに対し，10 枚では，エッジサーバ 5 台のときパイプラインシナリオ<並列処理シナリオ，10 台のときパイプラインシナリオ > 並列処理シナリオという関係になっており，さらに 15 枚では，共にパイプラインシナリオ < 分散処理シナリオという関係になっている．各条件における実行時間の値を用いて線形近似曲線を求めたところ，エッジサーバが 5 台のときは 8.79 枚，10 台のときは 12.02 枚のときに，式(3.11)の値が 0 になる，つまり，両シナリオの実行時間が等しくなるという結果になった．以上より，エッジサーバ上で分散される計算量が等しいアプリケーション処理を短時間で行いたい場合，まずはできるだけエッジサーバの台数が多い環境を用意することが求められる．その上で，処理回数が比較的少なく，かつ式(3.11)の値がプラスになるアプリケーションであれば並列処理シナリオを，処理回数が比較的多く，かつ式(3.11)の値がマイナスになるアプリケーションであればパイプラインシナリオを選択すべきであると考えられる．

次に，上記と同条件で実機評価を行い，理論値と比較することで遅延分析モデルの精度検証を行う．エッジ 1 台での処理実行時間を図 3.20 に，エッジサーバ数 5,10 台としたときのパイプラインシナリオと並列処理シナリオの処理実行遅延を図 3.21，3.22 にそれぞれ示す．

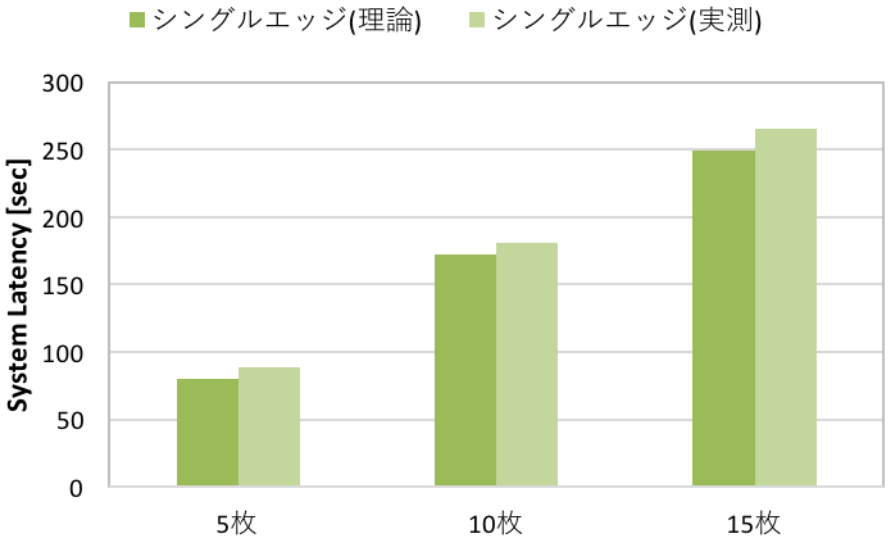

図 3.20 シングルエッジシナリオでの処理実行遅延特性 (n=1)(理論,実測値)

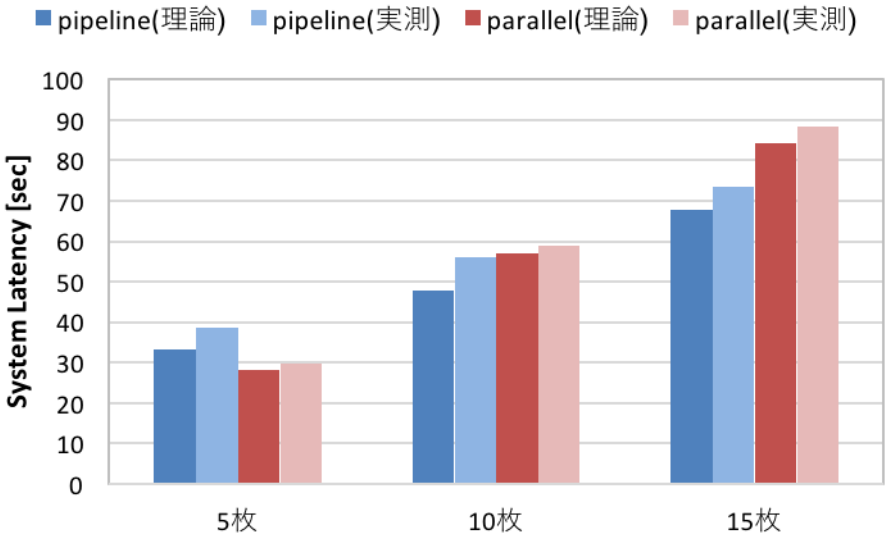

図 3.21 パイプライン・並列処理シナリオでの分散処理実行遅延比較（n=5)(理論,実測値)

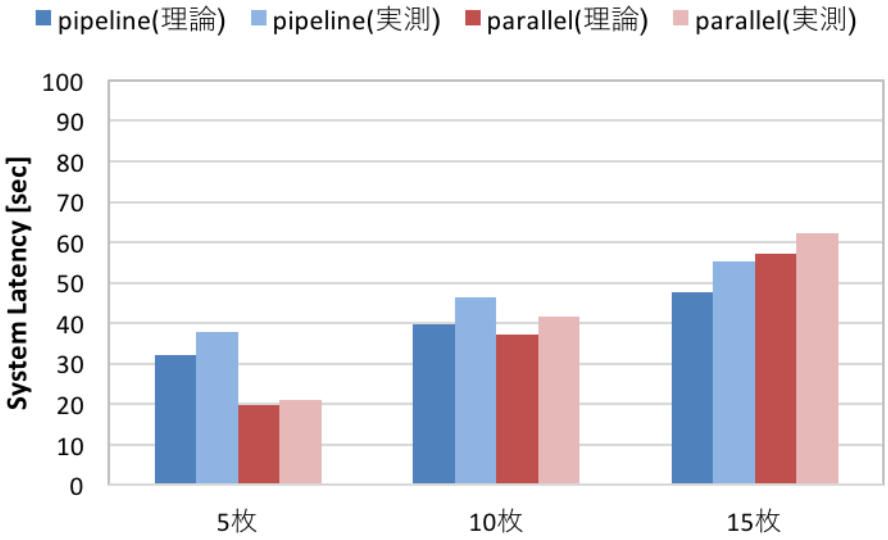

図 3.22 パイプライン・並列処理シナリオでの分散処理実行遅延比較（n=10)(理論,実測値)

図 3.20~3.22 より，生成画像の枚数が少ない 5 枚では，共に実行時間はパイプラインシナリオ > 分散処理シナリオという関係であるのに対し，10 枚では，エッジサーバ 5 台のときパイプラインシナリオ<並列処理シナリオ，10 台のときパイプラインシナリオ > 並列処理シナリオという関係になっており，さらに 15 枚では，共にパイプラインシナリオ < 分散処理シナリオという関係になっている．したがって，本環境下では実測値においても遅延分析モデルから導いた理論値同様の遅延特性を示すことが確認できた．なお，理論値よ

りも実測値が全体的にやや大きく，シングルエッジシナリオで約 7.0%，パイプラインシナリオで約 15.5%，並列処理シナリオで約 6.6%の誤差率となっている．

本節では，エッジコンピューティング環境でイメージモザイキング処理を行うことを想定し，生成するパノラマ画像の枚数を変化させたときの遅延特性を遅延分析モデルで評価し，実機評価の結果との比較を行った．本アプリケーションのような分散される計算量が等しいアプリケーションの実行においては，理論値・実測値ともに，多少の誤差率を伴うものの，同様の遅延特性を示すことが確認でき，遅延分析モデルの有効性が示された．

3.2.4 実機評価 2

本節では, エッジコンピューティング環境で 3.2.1 節 2)で紹介した赤外線熱画像を用いた可視光画像の人物検出を分散処理として行うことを想定し，人物検出する画像の枚数を変化させたときの遅延特性を遅延分析モデルで評価し，実機実験の結果と比較する．なお，赤外線熱画像を用いた可視光画像の人物検出は 3 つの処理から構成されており, 計算量がO_3 >>O_1 , O_2 といったように，遅延のボトルネックとなるような処理を含んでいる．

図 3.23，3.24 に実験環境を示す．まず，人物を可視光カメラと赤外線カメラで撮影した画像をエッジサーバに転送する．エッジサーバでは，可視光画像のサイズ調整を行い(処理 1 とする)，可視光画像と赤外線熱画像の位置合わせを透視投影変換で行う(処理 2 とする)．次に，HOG 特徴量を用いて可視光画像の人物検出を行い，この検出結果に対して温度情報による判定を行う(処理 3 とする)．最後に，判定結果が反映された人物検出画像をローカル端末に転送する．使用するエッジサーバは 3 台とし，各シナリオでのアプリケーション実行時間を，検出画像を 3,6,9,12 枚とした計 4 パターンで評価する．

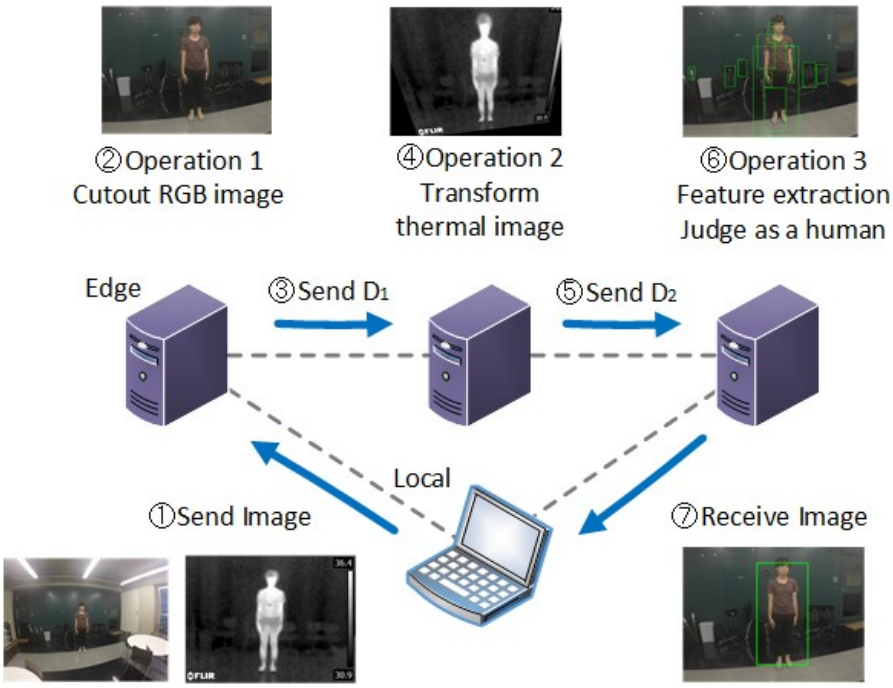

図 3.23 パイプラインシナリオでの実験環境

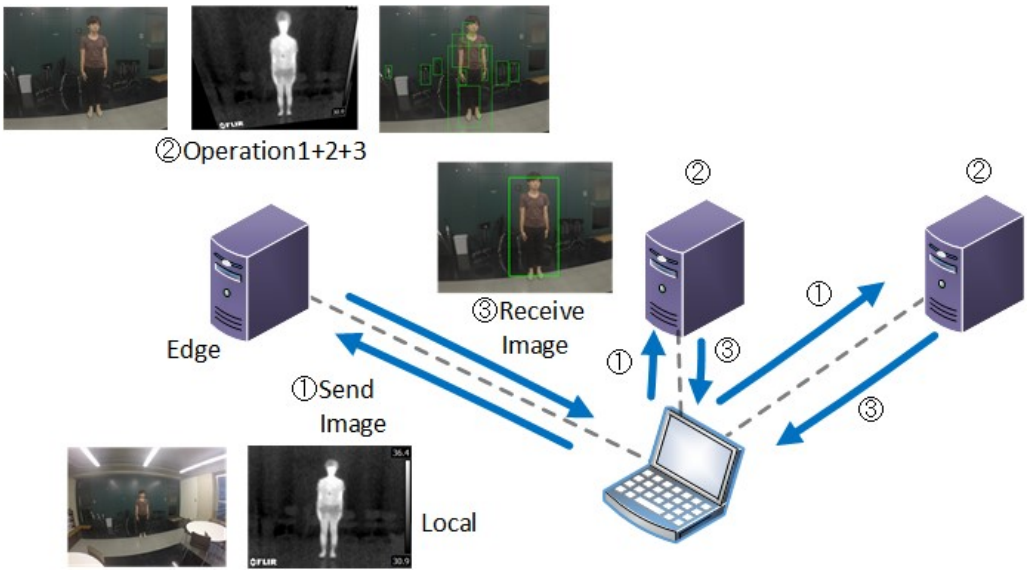

図 3.24 並列処理シナリオでの実験環境

次に，上記で設定した 4 パターンにおける，パイプラインシナリオ，並列処理シナリオでの分散処理実行遅延の理論値を図 3.25 に示す.

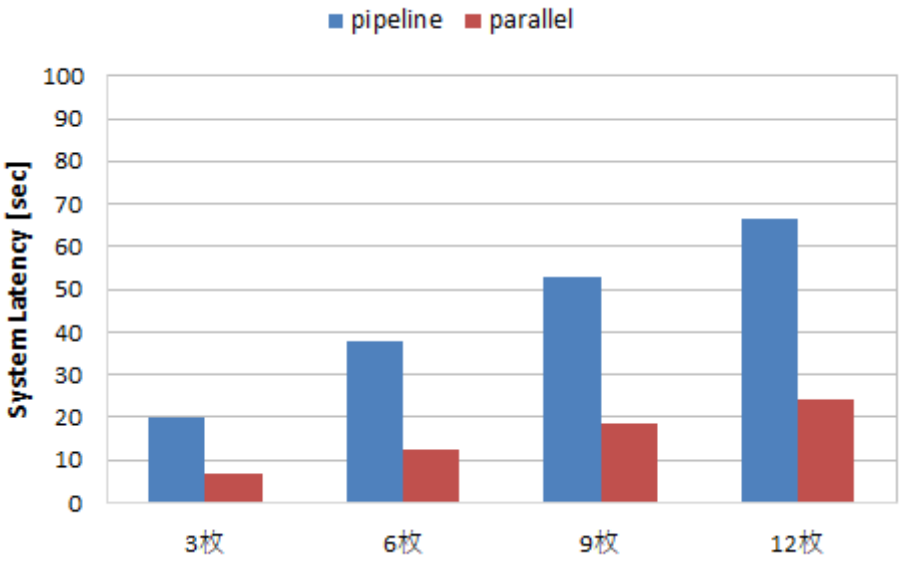

図 3.25 パイプライン・並列処理シナリオでの分散処理実行時間比較(理論値)

図 3.25 より，本環境下では理論的に，並列処理シナリオでいずれの枚数においてもパイプラインシナリオより短い時間での処理が可能であることがわかる．3.2.3 節の分散処理の評価で，パイプラインシナリオと並列処理シナリオの選択に関して，短時間でエッジサーバを活用したアプリケーション処理を行いたい場合，処理回数が少ないであれば並列処理シナリオを，ある程度の処理回数を要するアプリケーションであればパイプラインシナリオを選択すべきであると考察した．本環境ではエッジサーバの台数と処理回数が少ないため，並列処理シナリオのほうが短時間での処理が可能であるという理論値の結果は，3.2.3 節の考察に従っているといえる．また，イメージモザイキングでは計算量が等分されているのに対し，本アプリケーションでは処理 3 の処理時間が他の処理，通信時間との差が大きい($O_3 >> O_1, O_2$)ため，式(3.11)における各処理時間とデータ送信時間の最大値の蓄積分が大きくなることがパイプラインシナリオにおいて不利になると考えられる．

次に，パイプラインシナリオ，並列処理シナリオでの分散処理実行時間の理論値と実測値を並べたものを図 3.26 に示す．

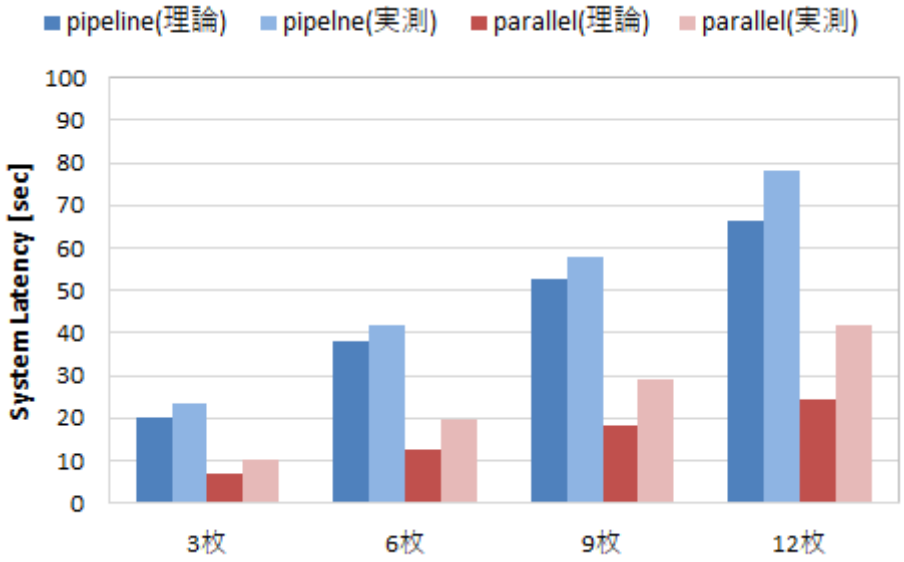

図 3.26 パイプライン・並列処理シナリオでの分散処理実行時間比較(理論,実測値)

図 3.26 より，理論値同様，本環境下では実測値からも，並列処理シナリオのほうが短時間での処理が可能であることがわかる．しかし，検出画像枚数が増えるほど理論値と実測値の差が大きくなっており，パイプラインシナリオで約 13.2%，並列処理シナリオで約 53.7%と，特に並列処理シナリオで誤差率が顕著となっている．検出画像をエッジサーバからユーザ端末に転送する際，ユーザ端末が別のサーバと通信している場合に通信経路の確保ができず，待機時間が発生していること，並列処理シナリオにおいてローカル端末から各エッジサーバに画像を送信する際のサーバ接続の切り替えが多く発生することなどが原因として考えられる．

本節では，赤外線熱画像を用いた可視光画像の人物検出をパイプラインシナリオ，並列処理シナリオに適用してアプリケーション実行時間を計測し，理論値との比較を行った．本アプリケーションのような遅延のボトルネックとなるような処理を含んでいるアプリケーションの実行においても，いずれも 3.2.3 節の考察内容に沿う結果が確認でき，遅延分析モデルの有効性が一部示された．しかし，実機実験により，遅延分析モデル内で考慮していなかったパラメータによる遅延が評価に影響を与えていること，遅延分析モデルで想定していた挙動を一部再現できていなかったことがわかったため，より誤差率の少ないより高精度な分析を行うためには，この 2 点を踏まえた遅延分析モデルの再検討が必要である．

3.3 OpenFlow を用いた経路設定

本節では，3.1 節で紹介した，従来のクラウドコンピューティング環境とエッジコンピューティング環境の特性を評価するための遅延分析モデルを利用して，処理環境におけるネットワークの使用状況や計算資源を考慮した上で最も低遅延処理が可能となる経路を検討し，実機実験による結果と比較して，遅延分析モデルおよび選択された最適経路の妥当性を評価する．2.3.1 節で紹介した，ソフトウェアによってネットワークの経路選択やデータ転送を柔軟に制御する SDN を実現する技術である OpenFlow[14]を用いて，エッジコンピューティング環境でアプリケーション処理を行う際の低遅延処理実現のためのネットワーク経路を行う．なお，OpenFlow を構成する環境として，OpenFlow スイッチを仮想ソフトウェアスイッチである OpenvSwitch[31]で, OpenFlow コントローラを Python ベースのフレームワークである Ryu[32]で実装し, 最適なネットワーク経路導出と経路制御に利用する.

本節で想定するエッジコンピューティング環境は，n 台のエッジサーバと OpenFlow コントローラで構成されており，m 回の処理が必要なアプリケーションに対して順に階層処理を行うものとする. OpenFlow コントローラはエッジコンピューティング環境のネットワーク状況や計算資源に関するリソース収集用として設置されており, OpenFlow スイッチである OpenvSwitch や各サーバから定期的に情報収集している．なお，単純化のため，OpenFlow コントローラは予め計測したアプリケーション処理に関わる一部のパラメータについての情報を把握しているものとする．このような環境において，次の手順でアプリケーション処理を実行する.

1) ネットワーク資源，計算資源情報およびアプリケーション処理に関するパラメータを所持している OpenFlow コントローラが，それらの情報からアプリケーション処理の通信コストと計算コストを見積もり，その収集時点で最も低遅延処理できる経路を決定する．なお，その情報は各エッジサーバに定期的に送信される.

2) アプリケーション処理に必要なデータ D_{rx} をユーザが用意し，1 台目のエッジサーバがそれを受信する．エッジサーバは，データを受信した時点で，コントローラから取得している経路情報に基づいて処理もしくは次のサーバへの送信を繰り返していき，m 回の処理後に生成した処理結果データ D_{tx} をユーザへ転送する.

次に, OpenFlow コントローラによる低遅延経路決定方法について説明する. エッジコンピューティング環境を，エッジサーバの集合 V={1,2…,n}とサーバ区間の集合 E={(1,2),(1,3),(1,4),(2,1)…}を持つ有向グラフ G = (V,E)と考え，この時のアプリケーション

処理の集合 M={1,2…,m}とすると，0-1 整数計画問題として次の目的関数，制約条件として設定できる．

Minimize:

$$\sum_{i,j\in V,k\in M}\left(\frac{O_k}{C_i}*\alpha_{i,k}+\frac{D_k}{bw_{ij}}*\beta_{i,j,k}\right) \quad (3.12)$$

Subject to:

$$\alpha_{i,k},\beta_{i,k}\ \in\{0,1\},(i,j\in V,k\in M) \quad (3.13)$$

$$\sum_{i,j\in V,k\in M}\alpha_{i,k}=m \quad (3.14)$$

$$\sum_{i,j\in V}\alpha_{i,k}=1,(k\in M) \quad (3.15)$$

$$\sum_{i,j\in V}\beta_{i,j,k}\begin{cases}\geq 1\ (i\neq j)\\ =0\ (i=j)\end{cases},(k\in M) \quad (3.16)$$

式(3.12)において，エッジサーバの処理コストとして処理時間を表す第一項と，サーバ区間の通信コストとして通信時間を表す第二項の総和を最小化するα, βを設定することで, 遅延を最小化する経路決定を行う．ここで，処理プロセス数 O, 処理速度 C, データ D, サーバ i から j へのスループット bw_{ij}は定数として扱うこととする．式(3.13)~(3.16)は制約条件を示す．$\alpha_{i,j,k}$,$\beta_{i,j,k}$は 0 or 1 をとる二値変数とし，アプリケーションの処理 k を実行する上で，サーバ i,j 区間を経由する，または，その経由したサーバ j で処理する場合の値は 1，経由しない, または処理しない場合の値は 0 とすると, この制約条件は式(3.13)で表現できる．さらに，処理 k は合計で必ず m 回発生するため，式(3.14)で表現でき，その中で各処理は必ず一度しか発生しないため，式(3.15)で表現できる．なお，処理 k は，同一サーバ(i=j)でも処理でき，その際は，通信は発生しない（$\beta_{i,j,k}$=0）．一方，処理 k は，同一サーバで処理しない場合(i≠j)は，必ずいずれかの処理サーバで処理する必要があるため，通信が発生する($\beta_{i,j,k}$=1)．2 ホップ以上跨ぐことも想定すると，式(3.16)のように表現できる．

ここで，低遅延経路決定の例として，エッジサーバ 3 台(n=3)，2 回 (m=2)の階層処理を行う場合を示す．図 3.27 のような環境において，2 回の階層処理を行うデータを，まずは①(i=1)のエッジサーバが受信する．本環境は，サーバの処理速度が③>①>②という関係にあり，①②サーバ間のスループット(bw$_{12}$, bw$_{21}$)が最も小さいため，OpenFlow コントローラは①，③サーバを通る経路で処理すれば最も低遅延な処理が可能になると算出できる．その算出結果を①サーバが受信し, OpenFlow コントローラで制御された経路上で処理を実行する．このときの遅延時間は以下のような式(3.17)で算出できる．

$$L_{edge} = \frac{O_1}{C_1} + \frac{D_1}{bw_{13}} + \frac{O_2}{C_3} + \frac{D_{rx} + D_{tx}}{bw_{edge}} \tag{3.17}$$

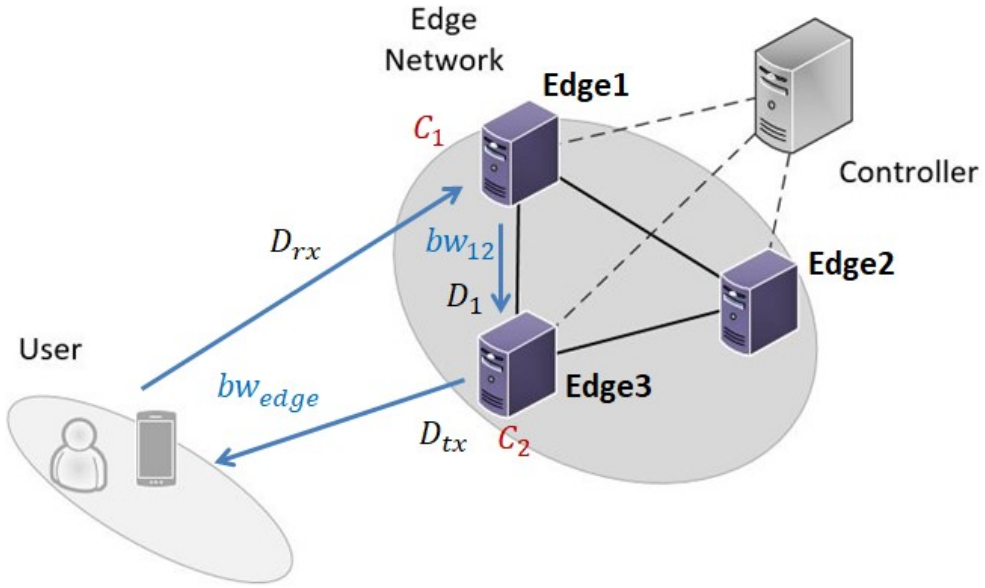

図 3.27 決定した処理経路例(n=3, m=2)

次に，図 3.28 に示すように，従来のクラウドコンピューティングおよびエッジコンピューティングを想定した実験環境を研究室内に構築し，アプリケーション処理の遅延特性評価を行う．本環境は，ユーザ端末 1 台とコアサーバ 1 台(Core)，エッジサーバ 3 台(Edge1,2,3)，OpenFlow コントローラ 1 台で構成されており，各サーバのスペックはそれぞれ表 3.3 の通りである．

表 3.3 コアサーバ，エッジサーバの概要

サーバ	リージョン	CPU	メモリ	OS
コアサーバ (Core)	東京	Intel® Xeon® E5-2650@2.30GHz	8GB	Ubuntu14.04
エッジサーバ (Edge1)	研究室内	Intel® Core™ i5-2400@3.10GHz	4GB	Ubuntu14.04
エッジサーバ (Edge2)	研究室内	Intel® Core™2 Q8400@2.66GHz	4GB	Ubuntu14.04
エッジサーバ (Edge3)	研究室内	Intel® Core™ i7-3770@3.40GHz	8GB	Ubutu14.04

ユーザ端末とコアまたはエッジネットワーク間は，[33]で想定しているクラウドコンピューティングの往復遅延が数百 ms，エッジコンピューティングの往復遅延が数十 ms 程度であることを踏まえ，それぞれ 100ms，10ms の遅延を発生させている．また，クラウドコンピューティングではエッジコンピューティングに比べて可用帯域が限られることを踏まえ，各区間の帯域幅をそれぞれ 10Mbps，100Mbps に制限している．遅延，帯域については，共に Linux のコマンドである tc (traffic control)コマンドで制御している．

OpenFlow コントローラは，上述のネットワーク経路設定手法に基づいて最適な経路を決定し，指定したサイクルごとに最適な経路情報を各サーバに対して送信している．各サーバはその経路に従ってデータ処理と転送を実行することとする．
本実験で想定するアプリケーションは，[34]で挙げられている監視カメラ映像からの人物検出である．アプリケーションの実行手順を次に示す．まず，監視カメラで撮影された映像を 2K の動画として 1 秒毎のセグメント(30 フレーム)ごとに分割し，コアもしくはエッジサーバへ送信する．次に，1 つ目の処理として，検出処理量緩和のために，動画のフレームサイズを 1920x1080 から 640x360 に縮小を行う．その後，2 つ目の処理として，各フレームに対して HOG 特徴量を用いて人物検出を行う．

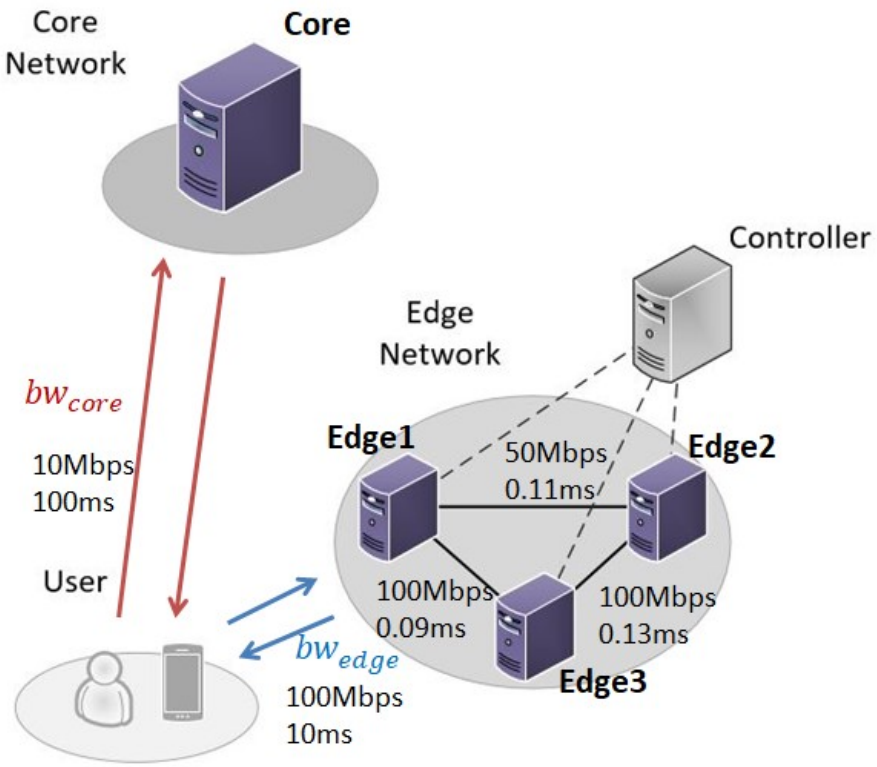

図 3.28 OpenFlow を利用した経路設定実験環境

ここからは，上述の人物検出アプリケーションについて，1 秒および 10 秒間の動画を処理したときの 3.1 節の遅延分析モデルに基づいて計算した実行遅延の理論値，および実機計測した実行遅延を図 3.29, 3.30 に示す．ここでは，コア，エッジサーバ(Edge1)それぞれで 2 つの処理を実行した結果と，式(3.12)~(3.16)の最適化問題に基づいて OpenFlow コントローラで決定した最適な経路での処理実行結果(Edge1→Edge3)を比較している．

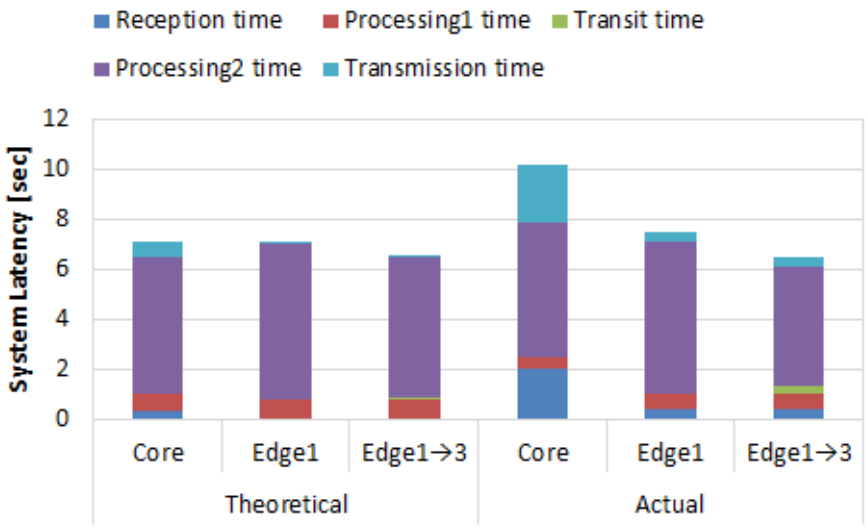

図 3.29 アプリケーション実行遅延理論値および実測値（元動画 1 秒）

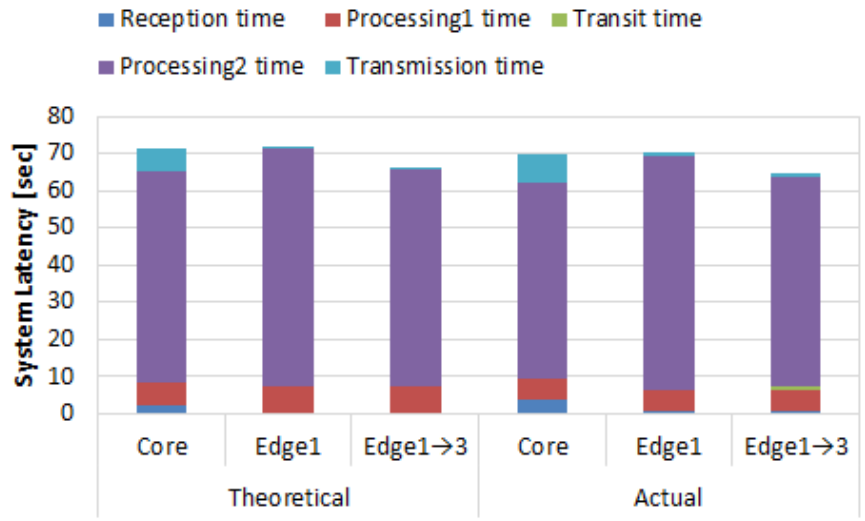

図 3.30 アプリケーション実行遅延理論値および実測値（元動画 10 秒）

図 3.29，3.30 からわかるように，処理時間(Processing1,2 time)に関しては理論値と実測値は概ね一致しているが，通信時間(Reception, Transit, Transmission time)に関しては，特にコアサーバで処理した際に約 3 秒程度の差が生まれている．この原因として, 今回計測した通信データサイズが, 図 3.29 の場合 D_{rx} =2.65[Mbit], D_{x} =5.52[Mbit] と小さく，TCP 輻輳制御が効果的に機能しなかったために理論値通りのスループットが出なかったことや，理論値に反映していないメッセージのやり取り時間が発生していることが原因として挙げられる．次に，エッジサーバとコアサーバを用いた際の通信にかかる時間を比較すると，エッジサーバを用いた場合の遅延時間は，いずれもコアサーバを用いた場合の遅延時間より短いことがわかる．これは主にユーザーコア/エッジサーバ間の帯域や Round Trip Time (RTT)といったネットワーク環境の差に起因しており，クラウドコンピューティングに比べてエッジコンピューティングは通信面において低遅延化を実現できることがわかる．一方，両サーバにおける処理にかかる時間を比較すると，今回のアプリケーションでは通信時間に比べて処理時間が支配的であり，たとえ通信時間がかかったとしても処理性能がより高いコアサーバで処理したほうが，通信時間と処理時間を合わせた総遅延時間がいずれ短くなるという結果が読み取れる．こういった特性はアプリケーション毎に異なるため，実際にアプリケーションを低遅延で処理したい場合，アプリケーション毎にアプリケーションと処理実行環境の各パラメータを把握し，式(3.6)のように通信量と処理量のバランスを考慮し，処理に利用するサーバを選択する必要がある．最後に，今回 OpenFlow コントローラにより，ネットワーク経路設定手法に基づいた低遅延処理のためのネットワーク経路最適化も行った．1 つ目の処理を Edge1，2 つ目の処理を Edge3 で行うという低遅延経路決定が行うことで，追加で発生する Edge1→Edge3 へのデータ転送時間を含めても，Edge1 で 2 つの処理を行うよりもトータルで実行遅延の低遅延化を達成できる．これより, OpenFlow コントローラによる処理環境の各リソース情報収集と低遅延処理のための経路制御は有効であるといえる．

本節では，エッジコンピューティングを用いたマルチメディア処理の低遅延化に着目し，モデル式や処理経路選択法の提案を行った．評価結果より，遅延分析モデルの妥当性と，クラウドコンピューティングおよびエッジコンピューティングの遅延特性を示した．さらに，モデル式を用いて低遅延処理のためにネットワーク経路を最適化することにより，エッジコンピューティング環境でより低遅延な分散処理が可能となることを示した．

第 4 章 仮想クラウド基盤を活用したエッジクラウドシステムにおけるマルチメディア処理実行遅延評価

本章では, エッジクラウドを想定した低遅延マルチメディア処理を行うためのエッジクラウドシステムを提案する. 図 4.1 のように, エッジコンピューティングでは, アプリケーション処理要求を行う端末とエッジ間の物理的な距離が同端末とクラウド間の距離に比べて大幅に短縮される. さらに, 複数エリアに分散配置されているエッジサーバで分散的にアプリケーション処理を実行できるため, サーバ 1 台あたりの処理負担も軽減される. しかし, それぞれのリソースが小規模なデータセンタのような構造で分散的に存在するため, 全体で見るとリソースの配置が複雑になってしまう. したがって, エッジコンピューティングを効率的に利用するために, 分散配置された計算リソースの中から, ユーザが要求する Quality of Service (QoS)を満たすように単一エリアのエッジサーバ内及び複数エリアのエッジサーバ間で使用するリソースを正しく選択する必要がある.

そこで, 提案システムにおいて, オーケストレータを用いたリソース操作やマルチメディア処理のスケジューリングのもと, アプリケーション処理を”機能”レベルで細分化し（スライシング), 連携する（チェイニング）ことで低遅延なアプリケーション実行環境を実現する. また, 仮想クラウド基盤技術として, オープンソースの OpenStack を利用することで, 効率的なリソース利用を図る. 実機評価より従来のクラウドよりも低遅延なアプリケーションを達成できることを示す.

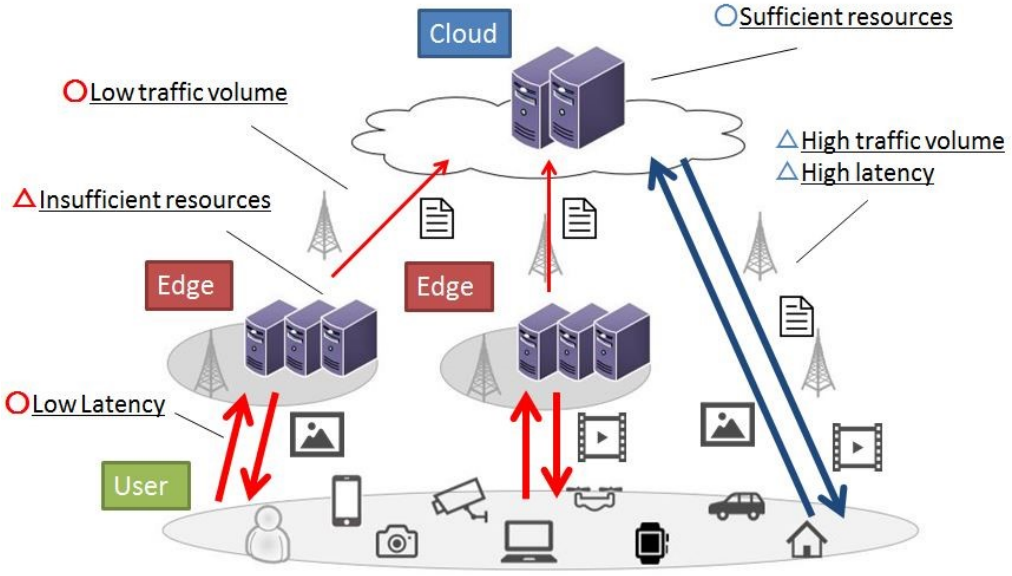

図 4.1 エッジコンピューティングと従来のクラウドコンピューティングの概念図

本章は文献[K. Imagane et al., “Performance Evaluations of Multimedia Service Function Chaining in Edge Clouds,” IEEE CCNC, Jan.2018.]に基づいている.

4.1 エッジコンピューティングのユースケースと課題

近年，上述の背景を受け，エッジコンピューティングに関する議論が盛んに行われている．その中で，エッジコンピューティングの特性を活かすためのシステムやアルゴリズム提案，評価に関連する研究が，主に想定されるエッジコンピューティングのユースケース別に行われている．

一つ目は，エッジコンピューティング環境を映像のエンコードやキャッシュのための映像配信プラットフォームとして利用するケースである[35][36]．計算リソースやキャッシュストレージの制約を考慮したアルゴリズムに関する研究が行われており，クラウドサーバと連携して，バックボーンのネットワークトラフィックや映像コンテンツの検索時間を最小化し，フリーズ時間の削減やレスポンス時間の向上といった映像配信の Quality of Experiment (QoE)の改善を図っている．

二つ目は，エッジコンピューティング環境内のエッジサーバをマルチメディア処理のためのオフロード先の対象として利用するケースである[37][38]．計算リソースの配置や利用スケジュールを最適化して，リアルタイムな並列分散処理を実現するための研究が行われている．[38]では，あらゆる端末で撮影された画像や動画に関する情報の検索・取得を，サーバと端末が連携した分散処理により実現するシステムの提案を行っており，エッジサーバへの処理のオフロードによるマルチメディア処理実行時間短縮と端末のバッテリー使用量削減が可能であることを示している．

三つ目は，エッジコンピューティング環境で映像監視システムをはじめとした IoT サービスを利用するケースである[39]．映像監視システムは，コンピュータビジョンやパターン認識などの複数の画像処理技術を統合しており，かつ，使用するデータや一部の処理を共有できるパターンも多い．そのため，データフュージョンや協調センシングといった分野に関して親和性が高いと考えられる．映像監視システムも含め，IoT サービスは全般的に高度な計算リソースの要求と遅延要求が発生するため, IoT により収集されたデータを，低遅延ネットワークを介してエッジコンピューティング環境に収集し，計算リソースを最大限に活用することが今後求められる．

これらに加え，他にもエッジコンピューティングのユースケースは多く存在するものの，いずれもリソースの”スケーラビリティ”と”マネジメント”という課題が挙がっている．エッジコンピューティングの計算リソースとして物理サーバを利用する場合は，ハードウェアの制約を受けるため，上記課題の解決が難しくなってしまう．また，仮想サーバを利用する場合は，クラウドプロバイダが提供する仮想サーバを用

いることで上記課題の解決にはつながるものの，設置場所がクラウドプロバイダのデータセンタに限られ，そこが必ずしもエッジコンピューティングで想定しているネットワークのエッジ部に該当するとはいえない．

4.2 エッジクラウドシステム

これまで，エッジコンピューティングにおける低遅延処理の実現について，第 3 章で遅延分析モデルを用いた遅延評価や実機実験を通して検証を行った．本章では，さらに仮想クラウドシステムという観点から，効率的なリソース利用を実現するために，OpenStack を活用したエッジクラウドシステムを提案する．その際，アプリケーション処理を「機能」レベルに分割し，エッジサーバ間において，機能や処理データも含め共有，再利用することで，並列分散処理によるアプリケーション処理の実行遅延の削減を図る．

4.2.1 システム概要

効率的なリソース利用を実現するためのエッジクラウドシステムの主な特徴をまとめる．

1) OpenStack を用いたエッジコンピューティング環境

物理的な構成にとらわれない，論理的なリソースの統合管理・運用を実現するために，IaaS 環境を構築するオープンソースのクラウド管理ソフトウェア群である OpenStack を用いてエッジコンピューティング環境(=エッジクラウド)を構築する．これにより，必要なリソースのスケーリングを動的に行うことが可能となる．

2) 複数エリア間でのマルチメディア処理機能の共有やデータ再利用

エッジコンピューティングの複数エリアの情報を収集するオーケストレータを設ける．情報収集と各処理のスケジューリングを行い，単一エリアのエッジサーバ内および複数エリアのエッジサーバ間で処理機能の共有や処理に必要なデータの再利用により，システム内のリソース利用効率を向上させる．

3) マルチメディア処理のためのサービススライシングとチェイニング

マルチメディア処理に必要な処理機能を細分化し，専用の仮想マシン(=インスタンス)として生成する(=サービススライスの生成)．さらに適切なスケジューリングにより，必要な機能を動的かつ適切に選択し，連携させる(=サービスチェイニング)．

4.2.2 エッジクラウド

本章で評価するエッジクラウドシステムの全体構成を図 4.2 に示す．また，各構成要素の詳細を以下にまとめる．

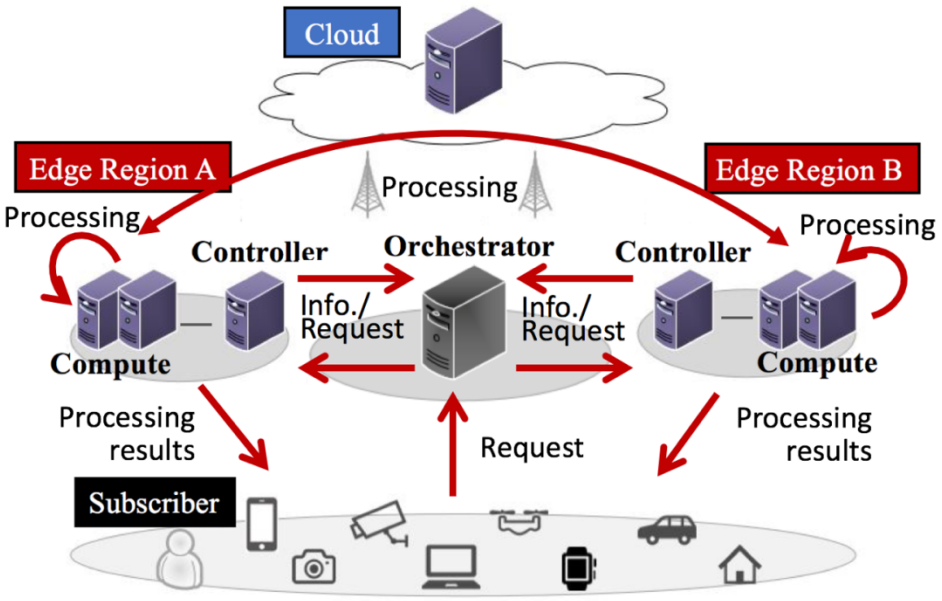

図 4.2 エッジクラウドシステムの全体構成

1) ユーザ(=Subscriber)

マルチメディア処理を要求する人もしくは端末である．

2) クラウドコンピューティング環境(Cloud)

マルチメディア処理に必要なリソースをユーザの遠隔地で提供する．

3) エッジコンピューティング環境(Edge)

マルチメディア処理に必要な計算リソースをユーザの近くで提供する．本システムでは，OpenStack Ocata（2017 年 2 月にリリースされた OpenStack のバージョン，詳細は 2.4.2 節)で構築された複数リージョンのクラウド環境となっており，以後，このエッジコンピューティング環境をエッジクラウドとして定義する．なお，各リージョンは 1 台のコントローラノードと複数台のコンピュートノードから構成されており，その中で複数のコンポーネント同士が図 4.3 のような連携をとっている．例として，2 リージョン構成(A,B)でコントローラノード 1 台, コンピュートノード 1 台の時の OpenStack 環境構成を図 4.4 に示す．

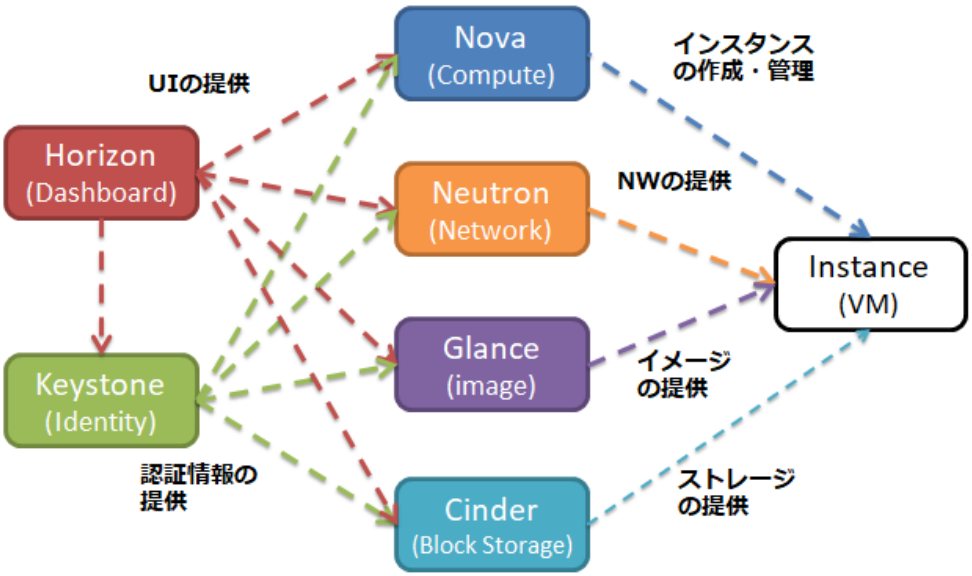

図 4.3 OpenStack コンポーネントの関係性

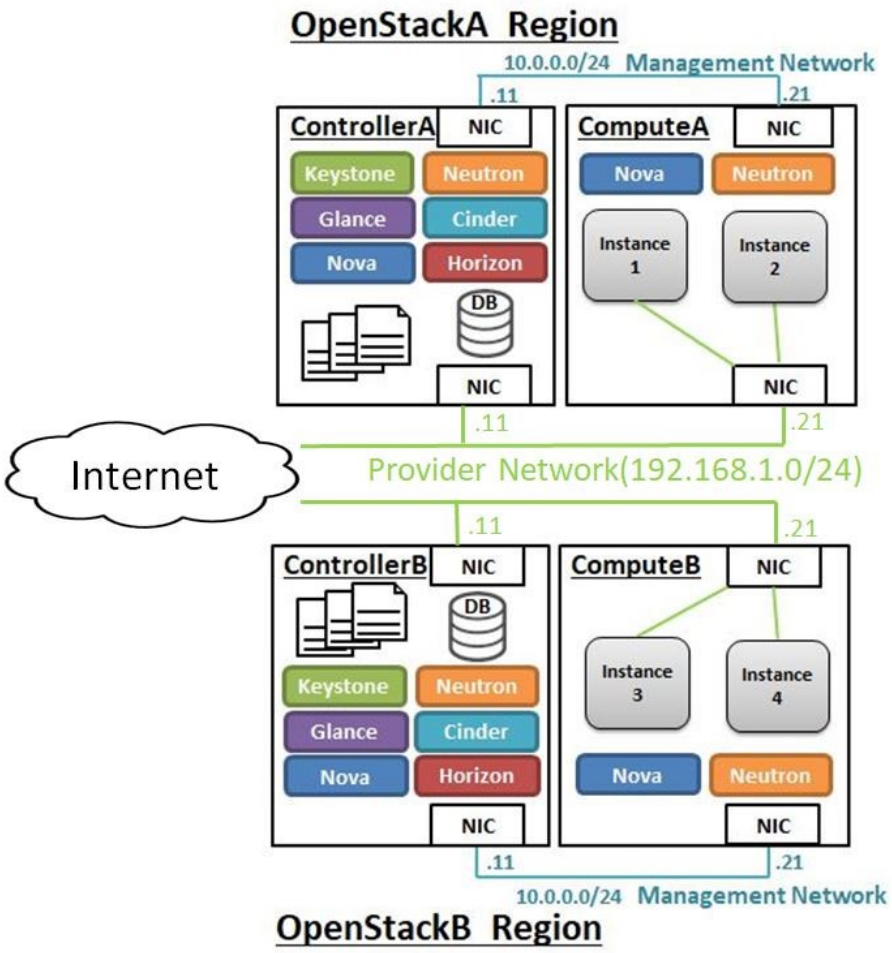

図 4.4 OpenStack 環境構成例

4) オーケストレータ

ユーザの要求情報や各リージョンのエッジクラウドリソースを把握し，エッジクラウドへのスケーリング指示やマルチメディア処理実行のスケジューリングを担う．詳細は 4.2.3 節に示す.

最後に，エッジクラウドシステムにおけるマルチメディア処理手順の概要を以下にまとめる.

1) オーケストレータおよび各リージョンのコントローラノードが定期的に各リージョンのリソース情報(ノード・リンク情報，アプリケーション情報)を取得する.

2) ユーザがオーケストレータに対してマルチメディア処理実行リクエストを送信する.

3) オーケストレータが 1,2 で得た 3 つの情報を基にマルチメディア処理の実行スケジューリングを決定する.

4) オーケストレータが該当のコントローラノードにスケジューリング結果を送信し，適切なエッジクラウドでマルチメディア処理を実行する.

5) 実行結果をユーザに送信する.

4.2.3 オーケストレータ

オーケストレータはエッジネットワーク内に配置され，OpenStack コントローラと合わせてエッジクラウドの各リージョンのノード・リンク情報(表 4.1) を定期的に収集している. また, アプリケーション情報(表 4.2)をあらかじめ与えられているものとし，ユーザの要求情報(表 4.3)はユーザのマルチメディア処理要求発生時に随時受け取るものとする．各情報は，OpenStack と親和性が高く，配列型や文字列型，数値など様々な形式のデータを扱いやすい JSON 形式でそれぞれリスト化する．また，それらの情報に基づいて，マルチメディア処理手順のスケジューリングを行う．スケジューリングの詳細については 4.2.6 節に示す.

本システムでは，マルチメディア処理機能を分割し，エッジクラウド内に各処理機能を搭載したインスタンスをそれぞれ生成し, ”マルチメディアサービススライシング”を行っている(詳細は 4.2.4 節)．そのため，アプリケーションサービスをユーザに提供するためには，それら複数の処理機能を連結する”マルチメディアサービスファンクシ

ョンチェイニング”を行う必要がある(詳細は 4.2.5 節)．また，既存の処理機能の連結だけでなく，使用予定のエッジクラウドの計算リソースが少ない場合や，対象リージョンが混み合っている場合などに，OpenStack の機能により，図 4.5 のように，リソーススケーリングやリソース複製といった，インスタンスへのリソース割り当てを可能とする．

表 4.1 エッジクラウド内のノード・リンク情報

JSON key	Type	Info.
node_info	Array	ノード情報
region	String	リージョン名
controllerIP	String	Controller の IP
instance	Array	インスタンス情報
function	String	処理機能
instanceIP	String	インスタンスの IP
time	Number	処理時間[s]
exe_history	String	App.実行履歴
link_info	Array	リンク情報
bw	Number	可用帯域[Mbps]

表 4.2 エッジクラウド内で実行するアプリケーション情報

JSON key	Type	Info.
app_info	Array	アプリケーション情報
name	String	アプリケーション名
procedure	Array	処理手順
function	String	処理機能
data	Number	データ量[MB]

表 4.3 エッジクラウドにアプリケーション処理要求を行うユーザ情報

JSON key	Type	Info.
user_info	Array	ユーザ情報
name	String	ユーザ名
region	String	近接リージョン名
application	String	アプリケーション名

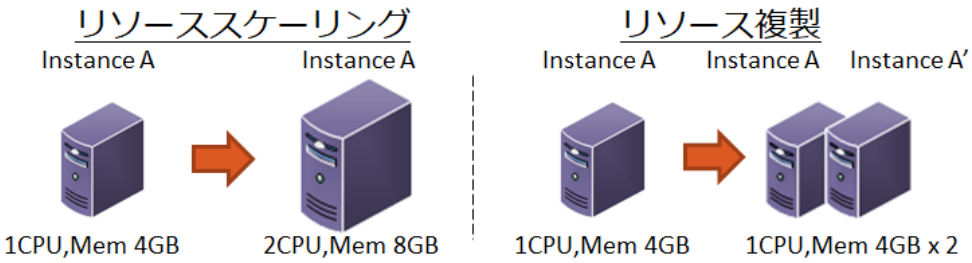

図 4.5 OpenStack のフレーバー更新機能によるインスタンスのリソース操作例

4.2.4 マルチメディアサービススライシング

マルチメディアサービススライシングでは，マルチメディア処理を機能レベルで定義・細分化する．さらに，エッジクラウド内にそれらの処理機能を搭載した専用のインスタンスを立ち上げ，処理結果やデータをサービス間で共有・再利用する．

ここで，本技術を人物検出処理に適用した場合の動作例を図 4.6 に示す．人物検出処理は，大きく「映像の取得」「エンコード」「検出処理」という 3 つの処理に分けることができる．マルチメディア処理のサービスは近年多様化しているものの，処理を分割した際の典型的な大枠はいずれも類似している．そのため，オーケストレータがこれらの共通した大枠を認識することで，複数のサービス間で処理機能を共有することが可能になり，効率的なリソース利用につながる．

なお，本研究では単純化のため，各インスタンスを一つの処理機能の専用マシンとし，処理機能の分割はアプリケーション実行前に行っておくものとする．マルチメディアサービススライシングを動的に行うためのアルゴリズムの検討と評価については，今後の課題とする．

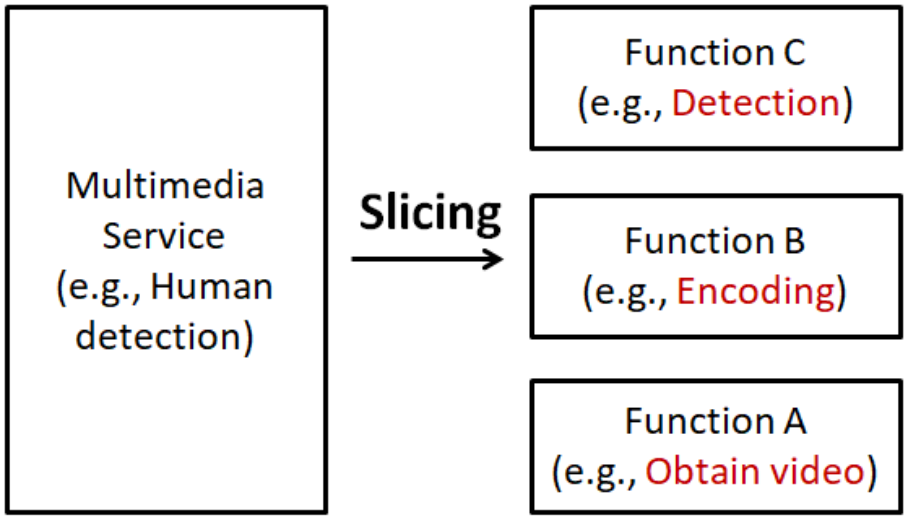

図 4.6 マルチメディアサービススライシング動作例

4.2.5 マルチメディアサービスファンクションチェイニング

エッジクラウドにおいて低遅延なマルチメディア処理を達成するためには，上記マルチメディアサービススライシングで分割されたサービスの処理機能を適切な順序，場所で提供する必要がある．そこで，マルチメディアサービスファンクションチェイニングでは，ユーザの要求とリソース情報に基づき，分割された処理機能を適切な順序，場所で提供し，連結することで，一つのマルチメディア処理という形で提供する．

ここで，本技術を人物検出処理に適用した場合の動作例を図 4.7 に示す．まず，ユーザがオーケストレータに対して人物検出処理を要求し，オーケストレータは人物検出処理をマルチメディアサービススライシングによって「映像の取得」「エンコード」「検出処理」という 3 つの処理に分ける．その上で，オーケストレータおよびコントローラノードで取得しているリソース情報に基づき，オーケストレータもしくはコントローラノード上で処理手順のスケジューリングを行う(詳細は 4.2.6 節)．この場合は，スケジューリングの結果，「映像の取得」を第一の処理としてリージョン A のインスタンスで実行，取得した映像を同じリージョン内の別のインスタンスへ転送した後，そのインスタンスで「エンコード」を第二の処理として実行，さらにそのエンコードした映像をリージョン B のインスタンスへ転送し，そのインスタンスで「検出処理」を第三の処理として実行，最後に処理結果をユーザに転送する，という手順で行うこととなる．

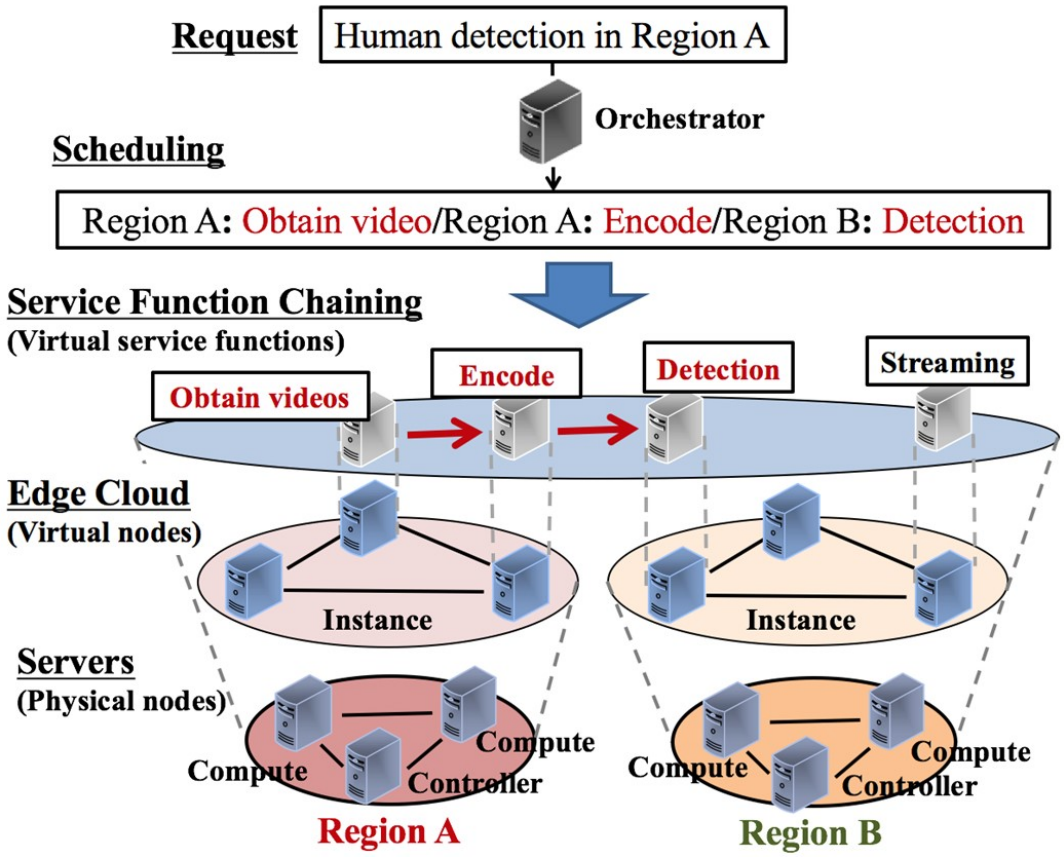

図 4.7 マルチメディアサービスファンクションチェイニング動作例

4.2.6 スケジューリングアルゴリズム

本システムにおいて，各マルチメディア処理のマルチメディアサービスファンクションチェイニング動作を決定するスケジューリングは，以下二つのアルゴリズム(Algorithm1，Algorithm2)によって決定される．

1) Algorithm1：各ユーザからの要求振り分け

本アルゴリズムでは，オーケストレータが各ユーザの要求を確認し，ユーザごとにどのリージョンのエッジクラウドでマルチメディア処理を実行するかを決定する．概要を表 4.4 に示す．

表 4.4 Algorithm1 概要

Algorithm 1 各ユーザからの要求振り分け	
Step1	アプリケーション要求として,ユーザ情報をオーケストレータが受け取る.
Step2	要求が残っていなければ,振り分けを終了する.
Step3	ノード・リンク情報/ユーザ情報/アプリケーション情報の 3 つの情報を照らし合わせる.
Step4	要求されたアプリケーションを実行中,もしくは実行結果をすでに持っているリージョンがあれば,そのリージョンのコントローラにユーザ情報を渡し,Step2 へ.
Step5	ユーザの近接リージョンのコントローラにユーザ情報を渡し,Step2 へ.

Step4 では，Step3 で参照したノード・リンク情報が持っている「exe_history：アプリケーションの実行履歴」とユーザ情報が持っている「name：アプリケーション名」，アプリケーション情報が持っている「function：処理機能」を照らし合わせ，一致する場合に該当のリージョンのコントローラの「controllerIP：コントローラの IP」をノード・リンク情報から取得し，ユーザ情報を送信する．既に実行中の要求と同じような要求が来た場合にその要求を実行しているリージョンに誘導することで，スライシングした処理機能の共有を図ることが可能になる．

2) Algorithm2：処理インスタンスの割り当て

本アルゴリズムでは，Algorithm1 の結果を受けてオーケストレータから受け取った情報をもとに，各リージョンのコントローラで，各処理を実行するインスタンスの割り当てとチェイニング動作を決定する．概要を表 4.5 に示す．

表 4.5 Algorithm2 概要

Algorithm 2 処理インスタンスの割り当て	
Step1	オーケストレータからユーザ情報を受け取る.
Step2	対応すべきユーザ情報がなければ割り当てを終了する.
Step3	ノード・リンク情報/ユーザ情報/アプリケーション情報の 3 つの情報を照らし合わせる.
Step4	同リージョン内での処理をベースとした最適化問題の解を導出する.リソースの共有を行う指示が出ている場合は,解にその指示を反映する.
Step5	最適化の結果を確認し，問題なければ Step7 へ.
Step6	同リージョン内で複数台のインスタンスによる処理,あるいは他のリージョンのリソースをオーケストレータに問い合わせて別のリージョンのインスタンスを組み合わせた処理を想定し,その候補となるインスタンスを決定する.
Step7	結果を順番通りにチェイニングし,リスト化してそれぞれのインスタンスで実行,Step2 へ.

Step4 では, 同リージョン内で各処理を実行するインスタンスの割り当てを決定するために最適化問題の解を導出する．スライスされた機能の集合を $P=\{1,2,\cdots,m\}$，インスタンスの集合を $I=\{1,2,\cdots,n\}$とし，0-1 整数計画問題として次の目的関数，制約条件を設定する．なお，$t_{i,j}$はインスタンス j における処理 i の処理時間とし，過去に該当の処理を実行した際の実測値をもとに，ノード・リンク情報の「time：処理時間」に格納されている．各インスタンスで処理できる機能と処理時間をノード・リンク情報から読み込み，問題を解くことで，どのインスタンスで何の処理を行うと最も低遅延で各処理を実行できるかを判断する．

Minimize:

$$\sum_{i\in P}\sum_{j,k\in I}\left(t_{i,j}*\alpha_{i,j}+\frac{D_i}{bw_{j,k}}*\beta_{i,j,k}\right) \quad (4.1)$$

Subject to:

$$\alpha_{i,k},\beta_{i,k}\ \in\{0,1\},(i\in P,\ j,k\in I) \quad (4.2)$$

$$\sum_{i\in P}\alpha_{i,j}\le 1,\ (j\in I) \quad (4.3)$$

$$\sum_{j\in I}\alpha_{i,j}\begin{cases}=0\ (\text{if } i \text{ can share})\\ =1\ (\text{else})\end{cases},(i\in P) \quad (4.4)$$

$$\sum_{j,k\in V}\beta_{i,j,k}\begin{cases}=0\ (\text{if } i \text{ can share})\\ =1\ (\text{else})\end{cases},(i\in P) \quad (4.5)$$

$$\alpha_{i,j}=\ \beta_{i,j,k}\ \ (k\in I) \quad (4.6)$$

$$\beta_{i,j,k}=0\ (\text{if } i=1 \text{ or } j=k),(i\in P,\ j,k\in I) \quad (4.7)$$

式(4.1)は，エッジクラウド内でのアプリケーション実行時間を算出する，本最適化問題の目的関数である．第一項がインスタンスの処理コストとして処理時間を，第二項がインスタンス間の通信コストとして通信時間を表しており，これらの組み合わせを最小化することを目的とする．また，制約条件を式(4.2)~(4.7)で 6 つ設定している．式(4.2)は 0(処理を行わない)または 1(行う)を判断する二値変数，式(4.3)は各インスタンスに対し処理は最大 1 つまでしか割り当てられないという制約．式(4.4)は各処理に対し，通常ならインスタンスが 1 台割り当てられ，処理の共有を行う際はインスタンスの割り当てを行わない，という分岐の制約. 式(4.5)は, 各処理に対してインスタンス間の通信が 1 回発生するという制約，式(4.6)は各処理を実行する場合，適切なインスタンスへ通信が行われ，実行しない場合はそのインスタンスが関連する通信は行われないという制約，式(4.7)は通信が発生しない条件を示している．

Step5 では, Step4 において同リージョン内で各処理につき 1 インスタンスを割り当てて実行するという想定のもと最適化問題を解き，リソース不足から遅延のボトルネックになるような処理を検知した場合，次の Step6 で該当の処理の低遅延化を図る．

Step6 では，同リージョン内のインスタンスのリソース操作(リソーススケーリングやリソース複製等)を行った上で，それらを用いた処理について検討する．また，他リージョンのリソース情報をオーケストレータに問い合わせ，リージョンを横断した処理についても

検討する．

Step7 では，アプリケーション情報の「procedure：処理手順」と Step6 までで決定したインスタンスと各処理の対応情報を参照し，正しい順番で該当のインスタンスの実行スクリプトを稼働する．

最後に，情報収集や各アルゴリズムの動作 1 回にかかる時間についての評価を行った．前提条件として，インスタンスが 5 台，処理機能が 3 つある場合，1 回の情報収集にかかる時間は，ノード・リンク情報で 2.17 秒，ユーザ/アプリケーション情報で 0.27 秒ほどであった．また，Algorithm1 は，各情報の送受信に 0.27 秒×2，各情報の参照に 0.03 秒，合計で 0.57 秒ほどであった．評価環境の制約はあるものの，いずれも簡単な JSON ファイルのやり取りと参照作業のみであるため，アプリケーション実行の全体遅延に影響を及ぼさない程度であることがわかる．一方，Algorithm2 は，各情報の送受信に 0.27 秒×2，各情報の参照に 0.03 秒，最適化問題の求解に 0.61 秒と，合計で 1.18 秒ほどであった．この最適化問題の求解時間に対し，インスタンス数が増加した場合の変化特性を掴むために，処理数 5 を固定としてインスタンス数が 1,5,10,15,20,25,30 の場合のデータセットを問題に与え，計算時間のシミュレーションを行った．結果を図 4.8 に示す．現状の問題設定では，インスタンス数が増加するにつれて求解時間が$O(n^2)$ で増加していくことが分かり，環境依存でアプリケーション実行の全体遅延に影響を及ぼさないようにするために，求解時間の発散を抑えるよう調整することを今後の課題とする．

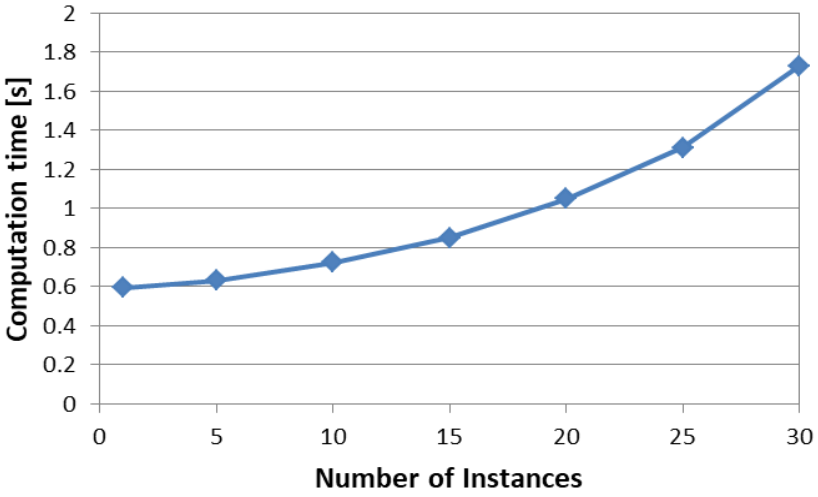

図 4.8 最適化問題の求解時間

4.3 遅延評価

本節では，4.2 節で説明したエッジクラウドを活用したマルチメディア処理システム上で複数のマルチメディアアプリケーションを実行した際の遅延特性を評価し，従来のクラウド環境と比較する．

4.3.1 実験環境

エッジクラウドを想定した実験環境を図 4.9 に示す．2 リージョンのエッジクラウド(Region A，B)を大学の 2 研究室内に OpenStack で構築し，コントローラノードを 1 台，コンピュートノードを 2 台ずつ設置する．また，オーケストレータを Region B に 1 台設置する．ここまでの各物理マシンのスペックは表 4.6 の通りである．また，ネットワークカメラを Region A に配置して，Region A 内で歩行している人物の様子を撮影するものとする．各コンピュートノード上には，処理機能別に，ネットワークカメラからの映像の取得機能を持つ”Camera”，動画像の変換機能を持つ FFmpeg [40]を実装した”FFmpeg”，機械学習による人物検出処理機能を持つ YOLOv2 [41]を実装した”YOLOv2”，適応レート制御による映像配信機能を持つ MPEG-DASH [42]を実装した”DASH”という計 4 台のインスタンスを生成し，これらを処理機能と定義する．また，クラウドコンピューティング環境には，東京都内にデータセンタをもつクラウドプロバイダが提供する仮想サーバ 1 台を利用し，エッジクラウドにある 4 つの機能を全て持っているものとする．各仮想インスタンスのスペックは表 4.7 の通りである．

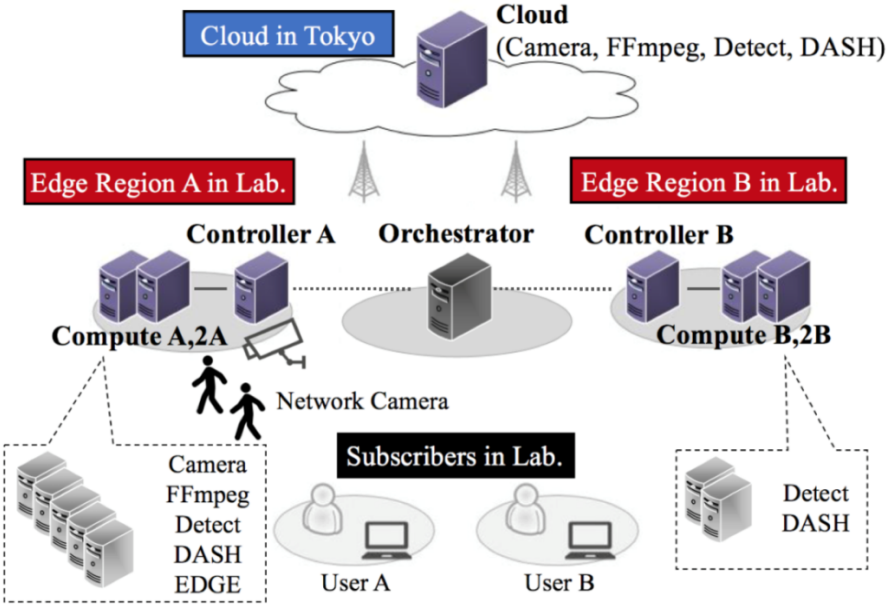

図 4.9 実験環境

表 4.6 物理サーバの概要

サーバ	リージョン	CPU	メモリ	OS
Compute A	リージョン A	Intel® Core™ i5-2400 @3.10GHz	8GB	Ubuntu16.04
Compute 2A	リージョン A	Intel® Core™ i7-6700 @3.40GHz	16GB	Ubuntu16.04
Compute B	リージョン B	Intel® Core™ i5-2400 @3.10GHz	16GB	Ubuntu16.04
Compute 2B	リージョン B	Intel® Core™ i5-3330 @2.70GHz	4GB	Ubuntu16.04
Orchestrator	リージョン B	Intel® Core™ i7-4770 @2.50GHz	16GB	Ubuntu16.04

表 4.7 仮想インスタンスの概要

インスタンス	リージョン	CPU 数	メモリ	ファンクション
Cloud	東京	20	224GB	ALL
Edge	リージョン A	2	4GB	ALL
Camera	リージョン B	2	4GB	CAMERA
FFmpeg	リージョン B	1	2GB	FFMPEG
Detect	リージョン A/B	2	4/8GB	DETECT
DASH	リージョン A/B	1	2GB	DASH

4.3.2 マルチメディアアプリケーション

本評価では，4.1 節で示したエッジコンピューティングのユースケースを元に，「映像監視システムによる人物検出」「MPEG-DASH による映像ストリーミング配信」の 2 つのアプリケーションを実行する．それぞれ概要を図 4.10，4.11 に示す．

1) Application1：映像監視システムによる人物検出

マルチメディアサービススライシングにより，「Camera」「FFmpeg」「Detect」の 3 つの処理機能に分割される．ユーザがアプリケーション実行を要求すると，一番目のインスタンス(Camera)がネットワークカメラから Region A 内の様子を映した映像を取得する．次に，二番目のインスタンス(FFmpeg)が取得した映像をエンコードする．

最後に，三番目のインスタンス(Detect)がエンコードされた映像に対して人物検出処理を行う．

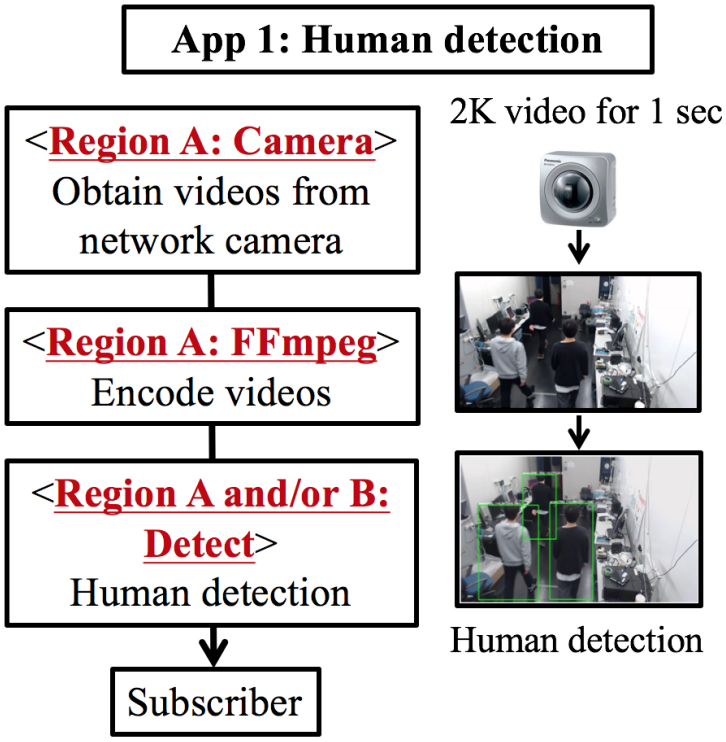

図 4.10 映像監視システムによる人物検出の概要

2) Application2：MPEG-DASH による映像ストリーミング配信

マルチメディアサービススライシングにより，「Camera」「FFmpeg」「DASH」の 3 つの処理機能に分割される．ユーザがアプリケーション実行を要求すると，一番目のインスタンス(Camera)がネットワークカメラから Region A 内の様子を映した映像を取得する．次に，二番目のインスタンス(FFmpeg)が取得した映像をエンコードする．なお，ここまではアプリケーション 1 と共通している．最後に，三番目のインスタンス(Detect)がエンコードされた映像を 4 つのビットレート(5Mbps，3Mbps，1Mbps，0.5Mbps)にトランスコードし，DASH ストリーミング用の Media Presentation Description (MPD)ファイルを作成する．

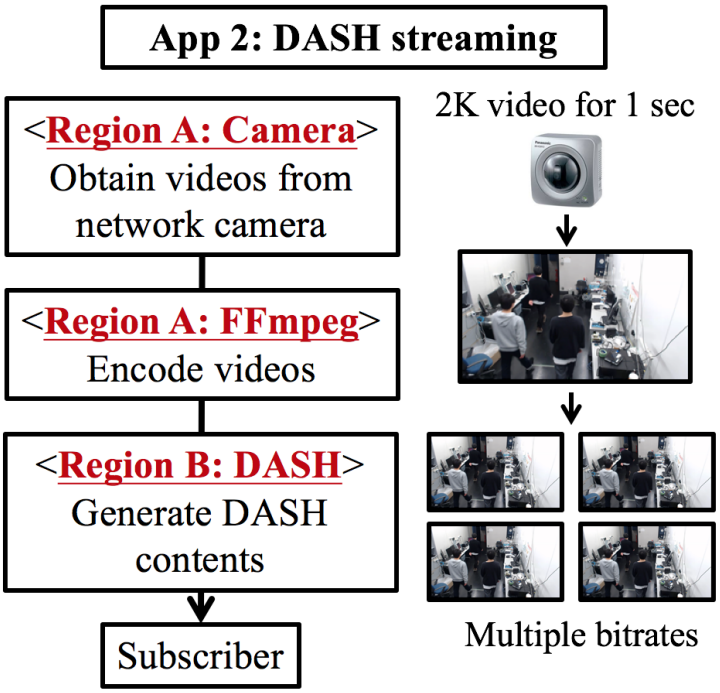

図 4.11 MPEG-DASH による映像ストリーミング配信の概要

4.3.3 実験シナリオ

本評価では，エッジクラウド内でのリソース操作やアプリケーション要求の組み合わせによる実行遅延の変化を評価する．評価するアプリケーションは 4.3.2 節で示した「映像監視システムによる人物検出」「MPEG-DASH による映像ストリーミング配信」とし，1 秒のセグメントの処理結果をユーザが 30 秒分要求する際の要求開始からセグメント到着までの時間を，以下 3 つのシナリオで評価する．

1) Scenario1：一人のユーザ(User A)が一つのアプリケーションを要求

処理を実行するインスタンス別に, シナリオを表 4.8 のように細分化する. 1-1 では，クラウドサーバ(Cloud)1 台でアプリケーション処理を実行する．1-2 では，エッジクラウドと同じリージョン内に存在するエッジサーバ(Edge)1 台でアプリケーション処理を実行する．1-3 では，エッジクラウド内の 3 台のインスタンスでスライシングされた 3 つの処理機能(Camera，FFmpeg，Detect/DASH)をそれぞれ実行する．ここで，オーケストレータの機能により，Application1 に関して，Detect 処理が遅延のボトルネックとなっていることを検知し，スケジューリングにより，1-4 ではメモリ量 2 倍のインスタンス(4GB→8GB)で Detect 処理を実行(リソーススケーリング)，1-5 ではこれま

で Detect 処理を実行していたインスタンスを複製し，2 台のインスタンスで Detect 処理を実行(リソース複製)する．

表 4.8 Scenario1 概要

シナリオ	ユーザ	インスタンス	リソース操作
1-1	A	Cloud	−
1-2	A	Edge	−
1-3	A	Camera->FFmpeg ->Detect/DASH	−
1-4	A	Camera->FFmpeg ->Detect*	メモリ増設 (Detect)
1-5	A	Camera->FFmpeg ->Detect*	複製 (Detect)

2) Scenario2：二人のユーザ(User A,B)が一つの同じアプリケーションを要求

処理を実行するインスタンスおよび要求するアプリケーション別に，シナリオを表 4.9 のように細分化する．Scenario1 同様，2-1 ではクラウドサーバ(Cloud)1 台で，2-2 ではエッジクラウドと同じリージョン内に存在するエッジサーバ(Edge)1 台でアプリケーション処理を実行する．2-3 も Scenario1 同様，エッジクラウド内の 3 台のインスタンスでスライシングされた 3 つの処理機能(Camera，FFmpeg，Detect/DASH)をそれぞれ実行するが，ここではオーケストレータの機能により，二人のユーザが同じアプリケーションを要求していることを検知し，User B に User A の実行結果を共有することで，両ユーザ通じてすべての処理を一度のみ行う(リソース共有)．なお，Detect 処理は 1-5 のようにリソース複製されたものを用いることとする．

表 4.9 Scenario2 概要

シナリオ	ユーザ	インスタンス	リソース操作
2-1	A	Cloud	−
2-1	B	Cloud	−
2-2	A	Edge	−
2-2	B	Edge	−
2-3	A	Camera->FFmpeg ->Detect*/DASH	−
2-3	B	Camera->FFmpeg ->Detect*/DASH	リソース共有 (ALL)

3) Scenario3：二人のユーザ(User A,B)が二つの異なるアプリケーションを要求

処理を実行するインスタンスおよび要求するアプリケーション別に，シナリオを表 4.10 のように細分化する．Scenario1 同様，3-1 ではクラウドサーバ(Cloud)1 台で，3-2 ではエッジクラウドと同じリージョン内に存在するエッジサーバ(Edge)1 台でアプリケーション処理を実行する．3-3 も Scenario1 同様，エッジクラウド内の 3 台のインスタンスでスライシングされた 3 つの処理機能(Camera，FFmpeg，Detect/DASH)をそれぞれ実行するが，ここではオーケストレータの機能により，二人のユーザが要求するアプリケーション間で共通した処理機能が存在すること(Camera，FFmpeg)を検知し，User B に User A の Camera，FFmpeg 実行結果を共有することで，User B は各アプリケーションの三番目の処理(Detect/DASH)のみを行う(リソース共有)．なお，Detect 処理は 1-5 のようにリソース複製されたものを用いることとする．

表 4.10 Scenario3 概要

シナリオ	ユーザ	インスタンス	リソース操作
3-1	A	Cloud	−
3-1	B	Cloud	−
3-2	A	Edge	−
3-2	B	Edge	−
3-3	A	Camera->FFmpeg ->Detect*	−
3-3	B	Camera->FFmpeg ->DASH	リソース共有 (Camera,FFmpeg)
3-3	A	Camera->FFmpeg ->DASH	−
3-3	B	Camera->FFmpeg ->Detect*	リソース共有 (Camera,FFmpeg)

4.3.4 実験結果および考察

本節では，各シナリオの実験結果を示す．各図の X 軸は，1 秒ごとのセグメントの番号，Y 軸は, ユーザが各セグメントの処理要求を開始からユーザに実行結果が到着するまでの時間を表しており，本節内で以降に出てくる”遅延”は，この”ユーザが各セグメントの処理要求を開始してからユーザに実行結果が到着するまでの時間”と定義する．

まず，一人のユーザ(User A)が一つのアプリケーションを要求する Scenario1 の結果を，

要求したアプリケーション別に，図 4.12，4.13 に示す．図 4.12 をみると，遅延が短い順に「クラウド 1 台<提案手法<エッジ 1 台」であることがわかる．これは，Application1 では Detect 処理に高度な計算資源を要するため，今回の評価環境でもっとも高度な計算資源を保有しているため(詳細なスペックは表 4.7 参照)，クラウドで最も低遅延処理が可能となっている．しかし，提案手法ではクラウドに比べて計算資源がかなり少ないにも関わらず，1-5 のようにリソースマネジメントを施すことで，クラウドの遅延に近い時間で処理が可能となっている．一方，図 4.13 をみると，遅延が短い順に「提案手法<エッジ 1 台<クラウド 1 台」であることがわかる．これは，Application2 では各処理に高度な計算資源を必要とせず，提案手法およびエッジ 1 台でエッジコンピューティングによる通信遅延の削減効果の影響で，クラウドに比べて低遅延処理を実現できているためである．

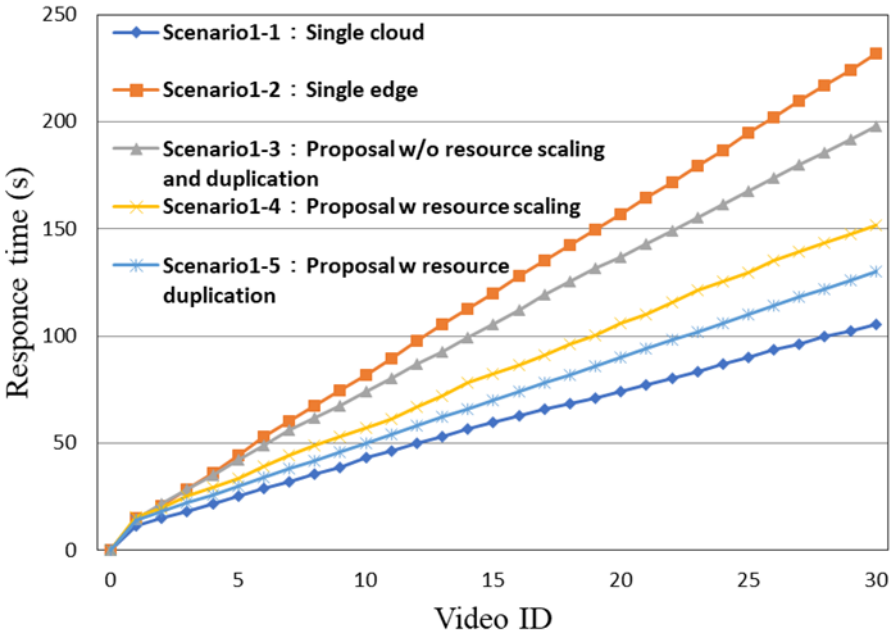

図 4.12 Scenario1 実験結果（Application1 要求時）

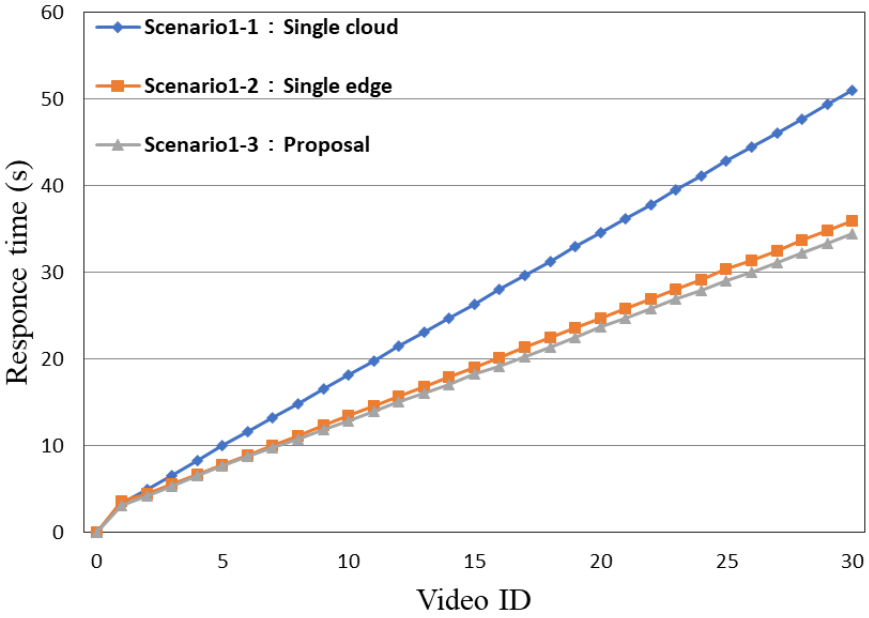

図 4.13 Scenario1 実験結果（Application2 要求時）

次に，二人のユーザ(User A，B)が同じアプリケーションを要求する Scenario2 の結果を，要求したアプリケーション別に示す．Application1 の結果を示す図 4.14 では，総遅延(2 ユーザのそれぞれの遅延の合計)が短い順に「提案手法<クラウド 1 台<<エッジ 1 台」，Application2 の結果を示す図 4.15 では，総遅延が短い順に，「提案手法<クラウド 1 台<<エッジ 1 台」であることがわかる．これは，2-3 の User B の処理では，オーケストレータおよびコントローラノードで User A の結果を共有してもらうよう指示を与えられており，両ユーザ通じてすべての処理が一度のみ行われるようになっていることが総遅延の削減につながっているためである．

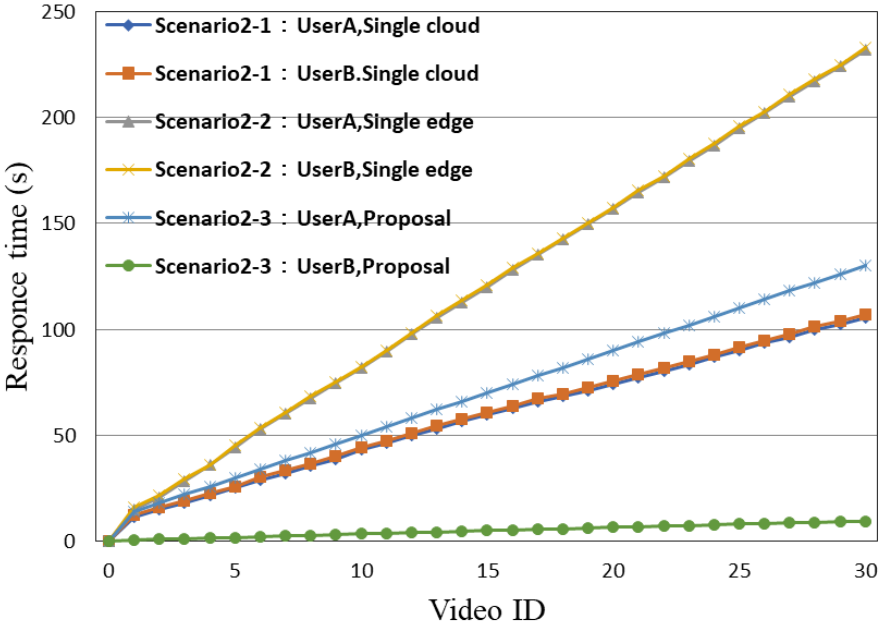

図 4.14 Scenario2 実験結果（Application1 要求時）

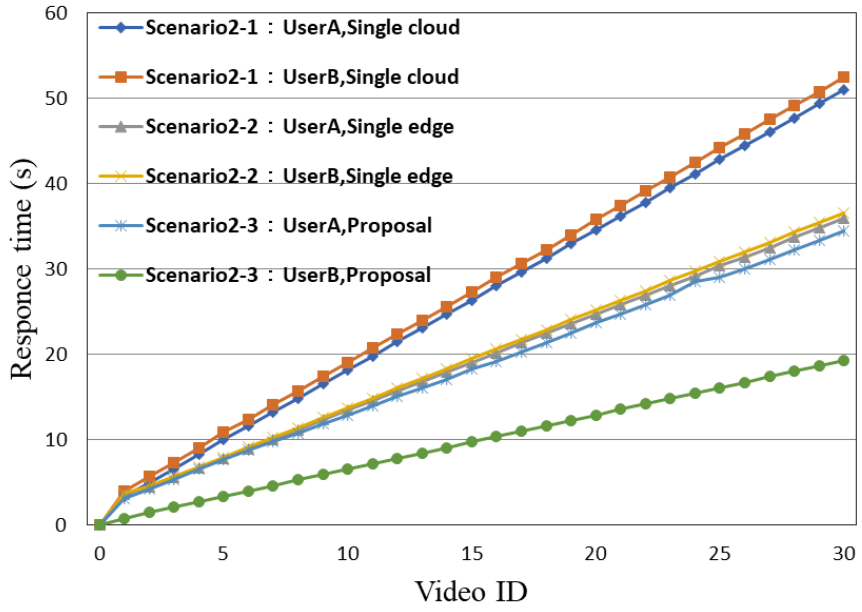

図 4.15 Scenario2 実験結果（Application2 要求時）

最後に，二人のユーザ(User A，B)が異なるアプリケーションを要求する Scenario3 の結果を，ユーザ別および要求するアプリケーションの組み合わせ別に示す．User A が Application1 を要求した結果を示す図 4.16 では，遅延が短い順に「クラウド 1 台<提案手法<エッジ 1 台」，Application2 を要求した結果を示す図 4.18 では，遅延が短い

順に「提案手法<エッジ 1 台<クラウド 1 台」となっているが，User A は Scenario1 の条件と変わらないため，Scenario1 と同様の結果となっている．一方，User B が Application 2 を要求した結果を示す図 4.17 では，遅延が短い順に，「提案手法<エッジ 1 台<クラウド 1 台」，Application 1 を要求した結果を示す図 4.19 では，遅延が短い順に，「提案手法<クラウド 1 台<エッジ 1 台」とであることがわかる．これは，3-3 の User B の処理では，オーケストレータおよびコントローラノードで User A の Camera, FFmpeg の処理結果を共有してもらうよう指示を与えられており，両ユーザ通じて Camera，FFmpeg の処理が一度のみ行われるようになっていること，その影響で，User B は Application1 要求時 Detect のみ, Application2 要求時 DASH のみ行われるようになっていることが，各遅延の削減につながっている．

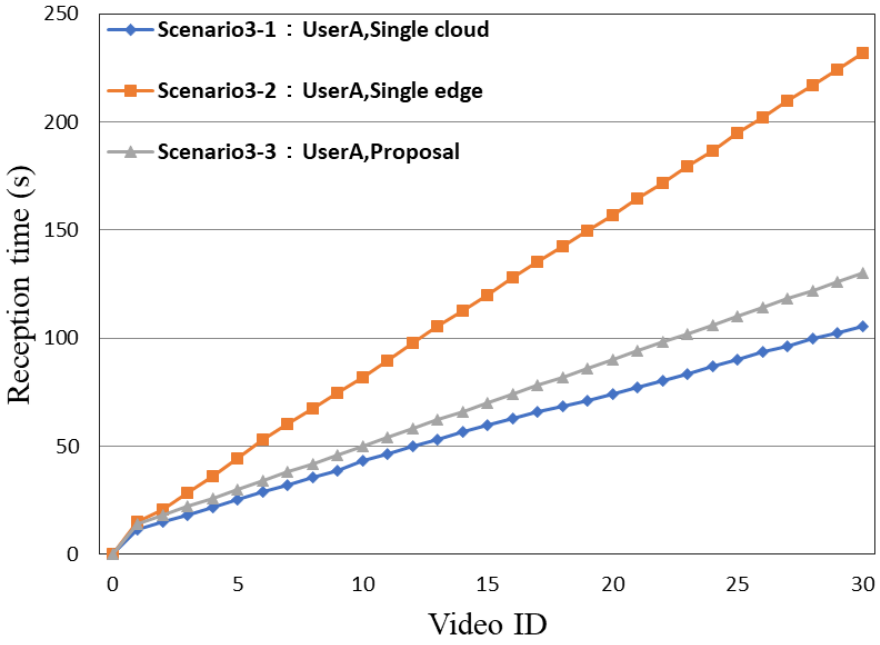

図 4.16 Scenario3 実験結果（User A のみ，User A:App.1 User B:App.2 要求時）

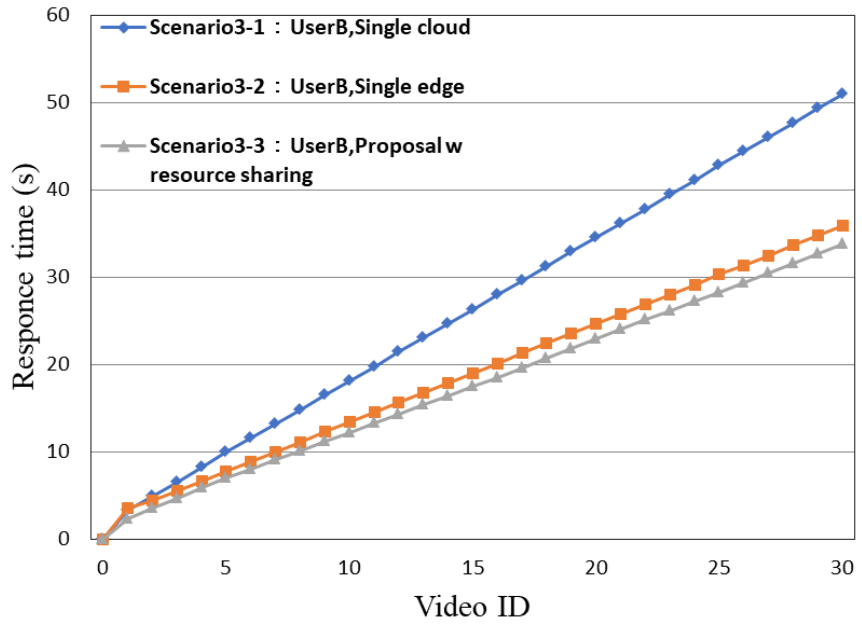

図 4.17 Scenario3 実験結果（User B のみ，User A:App.1 User B:App.2 要求時）

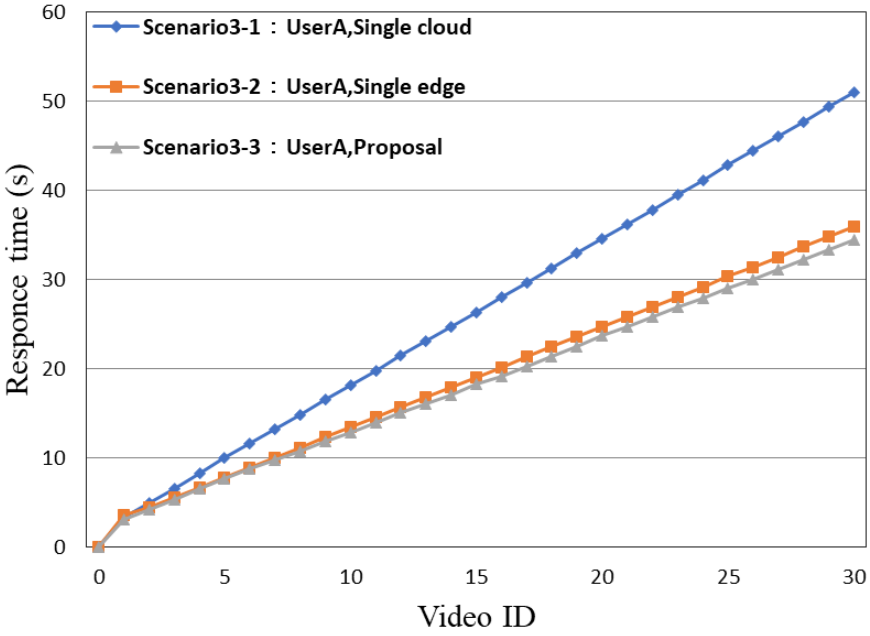

図 4.18 Scenario3 実験結果（User A のみ，User A:App.2 User B:App.1 要求時）

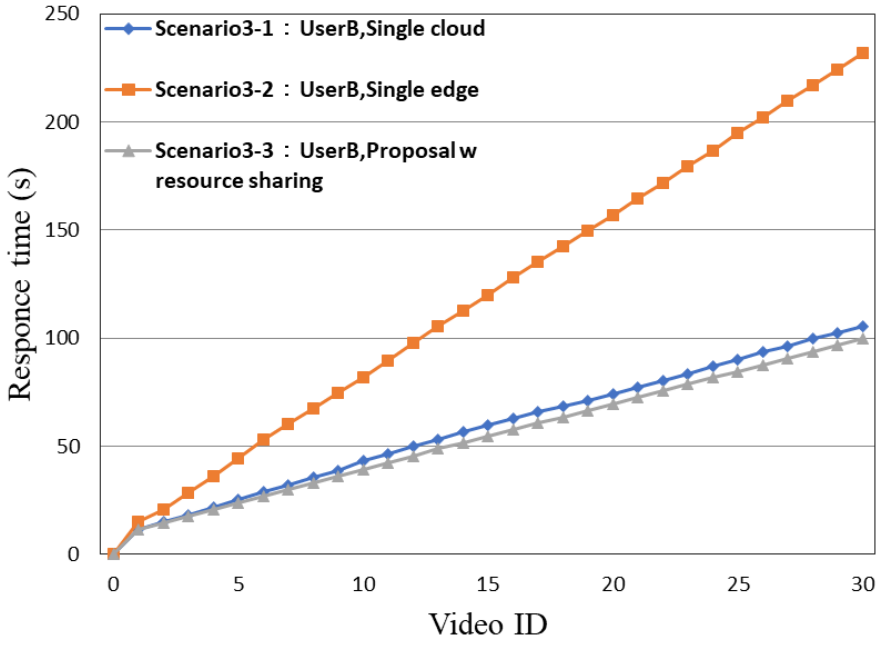

図 4.19 Scenario3 実験結果（User B のみ，User A:App.2 User B:App.1 要求時）

ここで，3 シナリオの結果をまとめる．「クラウド 1 台とエッジ 1 台」比較時は，高度な計算資源が必要な場合は，豊富な計算資源の活用により処理遅延を削減できるクラウドが有利であり，高度な計算資源を要しない場合は，通信遅延を削減できるエッジが優位になるといえる．「エッジ 1 台と提案手法」比較時は，提案手法においてエッジ 1 台利用時に比べ，計算資源が増えることはもちろん，適切なインスタンスのチェイニングによってサーバ 1 台あたりの処理負担の軽減が効果的に働く提案手法が優位になるといえる．「クラウド 1 台と提案手法」比較時は，提案手法において，クラウドに比べて計算資源が乏しいにも関わらず，様々なリソースマネジメントを施すことで，クラウドの遅延に近い時間，もしくはそれよりも短い時間で処理が可能になるといえる．

本章では，エッジクラウドを活用したマルチメディア処理システム上で複数のマルチメディアアプリケーションを実行した際の遅延特性を評価し，従来のクラウド環境との比較を行った．エッジクラウドを OpenStack を用いて構築し，オーケストレータと連携して，マルチメディアサービススライシング，マルチメディアサービスファンクションチェイニングといった技術や，ユーザの要求・実行環境に応じたリソース操作により，マルチメディア処理の機能共存やデータ再利用，並列分散などといったリソース操作を可能とするマルチメディア処理システムを実現した．その結果，本システム上においてアプリケーション実行遅延の削減が可能であることを示した．

第 5 章　総括

5.1 まとめ

本研究では，はじめに，従来のクラウドコンピューティングおよびエッジコンピューティング環境においてマルチメディア処理を実行する際の遅延特性をシナリオ別に理論モデル化し，それぞれの遅延分析を行った．また，同環境において，ネットワークの使用状況や計算資源を踏まえて，さらなる低遅延処理を実現するためのネットワーク経路の最適化を試みた．評価結果より，遅延分析モデルの妥当性と，クラウドコンピューティングおよびエッジコンピューティングの遅延特性を示した．さらに，モデル式を用いて，低遅延処理のために，ネットワーク経路の最適化によってエッジコンピューティング環境でより低遅延な分散処理が可能となることを確認した．

次に，ユーザが要求するマルチメディア処理を低遅延で実行するためのエッジクラウドシステムを提案し，システム上で複数の処理を実行した際の遅延特性を評価し，従来のクラウド環境との比較を行った．本システムは，仮想サーバ・ネットワークという観点から効率的なリソース利用を実現するために OpenStack を活用しており，オーケストレータと連携して，マルチメディアサービススライシング，マルチメディアサービスファンクションチェイニングといった技術や，ユーザの要求・実行環境に応じたリソース操作により，マルチメディア処理の機能共存やデータ再利用，並列分散などを可能とした．本システムに対して，研究室内およびクラウドプロバイダが提供するサーバ上に評価環境を構築し，複数シナリオにおける実機実験による遅延評価を行うことで，実行遅延削減の観点でシステムの有効性を確認した．

5.2 今後の展望

本研究で提案したマルチメディア処理の実行遅延の理論モデルにおいて，各パラメータの再検討を行い，より精度の高い理論値の算出と実機実験における理論モデルの再現性の向上を図る．また，マルチメディア処理を低遅延で実行するためのエッジクラウドシステムにおいて提案した技術の一つであるマルチメディアサービススライシングに関して，これを動的に行うためのアルゴリズムの検討を行い，他の技術と合わせてより大規模かつ複雑な実環境における提案システムの遅延評価を行う．

謝辞

本研究を行うにあたり，日頃から様々なご指導をしていただきました甲藤二郎教授に心から感謝の意を表します．また，研究を進める上で貴重なアドバイスをいただいた金井謙治助教をはじめ，研究だけでなく様々な面においてお世話になりました甲藤研究室の先輩，同期，後輩の皆様に深く御礼申し上げます．

最後に，ここまで育ててくださった両親に，深く感謝いたします．

2018 年 1 月 30 日

今金　健太郎

参考文献

[1]. Cisco, “Visual Networking Index: Global Mobile Data Traffic Forecast Update 2016-2021,” [online]: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf

[2]. ETSI “Multi-access Edge Computing,” [online]: http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[3]. ETSI “Cloud Computing,” [online]: http://www.etsi.org/technologies-clusters/technologies/cloud-computing

[4]. “第 3 の情報処理拠点を設けるエッジコンピューティング”, NTT COMWARE Corporate Magazine, vol. 61, pp.2-6, 2014.

[5]. Cisco, “Fog Computing ant the Internet of Things: Extended the Cloud to Where the Things Are,” [online]: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[6]. ZD Net Japan, “「フォグコンピューティングは定着するか」,” [online]: https://japan.zdnet .com/article/35084645/

[7]. W.Zhu, C.Luo, J.Wang, S.Li, “Multimedia Cloud Computing,” IEEE Signal Processing Magazine, vol. 28, issue: 3, pp. 59-69, May.2011.

[8]. Oracle VM VirtualBOX, [online]: http://www.oracle.com/technetwork/jp/server-storage/virtualbox/overview/index.html

[9]. VMware, [online]: https://www.vmware.com/jp/products/vsphere.html

[10]. Windows Virtual PC, [online]: https://www.microsoft.com/ja-jp/download/details.aspx?id=3702

[11]. Linux KVM, [online]: https://www.linux-kvm.org/page/Main_Page

[12]. XenServer, [online]: https://www.citrix.com/products/xenserver/

[13]. H. Kim, N. Feamster, “Improving Network Management with Software Defined Networking,” IEEE Communications Magazine, vol.51, no.2, pp.114-119, Feb. 2013.

[14]. OpenFlow, [online]: https://www.opennetworking.org/sdn-resources/openflow

[15]. ETSI GS NFV 002: “Network Functions Virtualization (NFV); Architectural Framework,”[online]: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf

[16]. A. M. Medhat, T. Taleb, A. Elmangoush et al., “Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges.” IEEE Communications Magazine, vol.55, no.2, pp.216-223, Feb.2017.

[17]. “サービスチェイニングについて IETF 新方式での 6 社相互接続を世界に先駆けて実証

~サービス識別タグを用いた転送で自由自在にサービスチェインを構成~," NTT 持株会社ニュースリリース, [online]: http://www.ntt.co.jp/news2015/1502/150212a.html
[18]. 中里秀則, クラウドシステム 講義資料:03-04. Oct.2015.
[19]. Amazon, AWS, [online]: https://aws.amazon.com/jp/
[20]. Google, Google Cloud Platform, [online]: https://cloud.google.com/
[21]. Microsoft, Microsoft Azure, [online]: https://azure.microsoft.com/ja-jp/
[22]. AWS Region and Endpoints, [online]: http://docs.aws.amazon.com/general/latest/gr/rande.html#ec2_region
[23]. Gartner Japan, "日本におけるクラウド・コンピューティングに関する調査結果," [online]: http://www.gartner.co.jp/press/html/pr20170404-01.html
[24]. OpenStack, [online]: https://www.openstack.org/
[25]. Apache CloudStack, [online]: https://cloudstack.apache.org/
[26]. 日本 OpenStack ユーザ会, "OpenStack クラウドインテグレーション," 翔泳社, 2015.
[27]. "OpenCV.jp", [online]. http://opencv.jp/.
[28]. 高橋沙季, 竹内健, 折橋翔太, 甲藤二郎, "赤外線熱画像を用いた可視光画像の人物検出精度改善の検討", 映像情報メディア学会冬季大会, Dec. 2015.
[29]. "Amazon EC2", [online]. http://aws.amazon.com/jp/ec2/.
[30]. "さくらのクラウド", [online]. http://cloud.sakura.ad.jp/.
[31]. OpenvSwitch, [online]: http://www.openvswitch.org/
[32]. Ryu, [online]: https://osrg.github.io/ryu/
[33]. 田中裕之, 高橋紀之, 川村龍太郎, "IoT 時代を拓くエッジコンピューティングの研究開発," NTT 技術ジャーナル, vol.27, no.8, pp.59-63, 2015 年 8 月.
[34]. 小川啓吾, 金井謙治, 竹内健, 甲藤二郎, 津田俊隆, "監視映像システムのための複数センサを活用したイベントドリブン型適応レート制御の性能評価", 電子情報通信学会 MoNA 研究会, 2016 年 10 月.
[35]. A. Lobzhanidze and W. Zheng, "Proactive caching of online video by mining mainstream media," in Proc. IEEE ICME 2013, pp.1-6, Jul.2013.
[36]. S.Islam and J.Gregorie, "Giving users an edge: A flexible Cloud model and its application for multimedia," Future Generation Computer System, pp.823-832, Jun.2012.
[37]. B.Girod, et al, "Mobile Visual Search," IEEE Signal Processing Magazine, vol.28, no.4, pp.61-76, Jul.2011.
[38]. S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," IEEE INFOCOM, pp.945-953, Mar.2012.

[39]. X. Wang, “Intelligent multi-camera video surveillance: a review,” Pattern Recognition Letters, Vol.34, Issue 1, pp. 3-19, Jan.2013.

[40]. FFmpeg, [online]: https://ffmpeg.org/

[41]. J. Redmon, S. Divvala, R. Girshick, A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788, Jun.2016.

[42]. I.Sodager, “The MPEG-DASH Standard for Multimedia Streaming Over the Internet,” IEEE Computer Society, vol.18, Issue4, pp.62-67, Apr.2011.

発表文献リスト

[1] 今金健太郎, 金井謙治, 甲藤二郎, “マルチメディア処理におけるエッジコンピューティングの特性評価,” 映像情報メディア学会冬季大会, 2015 年 12 月.

[2] 今金健太郎, 金井謙治, 甲藤二郎, 津田俊隆, “エッジコンピューティングにおける遅延分析モデルの一検討,” 電子情報通信学会総合大会, 2016 年 3 月.

[3] Kentaro Imagane, Kenji Kanai and Jiro Katto, “Performance Evaluations of Edge Computing for Multimedia Data Processing,” IEEE Smart Info-Media Systems in Asia (SISA) 2016, Sep.2016.

[4] Kentaro Imagane, Kenji Kanai, Jiro Katto and Toshitaka Tsuda, “Evaluation and Analysis of System Latency of Edge Computing for Multimedia Data Processing,” IEEE Global Conference on Consumer Electronics (GCCE) 2016, Oct.2016.

[5] 今金健太郎, 長島達哉, 金井謙治, 甲藤二郎, 津田俊隆, “エッジコンピューティングを活用したマルチメディア分散処理のための遅延評価,” 電子情報通信学会 通信方式(CS)研究会, 2016 年 12 月.

[6] 今金健太郎, 金井謙治, 甲藤二郎, 津田俊隆, “遅延分析モデルを用いたエッジコンピューティングとクラウドコンピューティングの実行遅延評価,” 電子情報通信学会総合大会, 2017 年 3 月.

[7] 今金健太郎, 金井謙治, 甲藤二郎, 津田俊隆, 中里秀則, “低遅延マルチメディア処理のための OpenStack を活用したエッジクラウドシステム,” 電子情報通信学会 通信方式(CS)研究会, 2017 年 7 月.

[8] Kenji Kanai, Kentaro Imagane and Jiro Katto, “Overview of Multimedia Mobile Edge Computing,” ITE Transactions on Media Technolgy and Applications, Vol.6, No.1, pp.46-52, Jan.2018.

[9] Kentaro Imagane, Kenji Kanai, Jiro Katto, Toshitaka Tsuda and Hidenori Nakazato, “Performance Evaluations of Multimedia Service Function Chaining in Edge Clouds,” IEEE Consumer Communications & Networking Conference (CCNC)2018, Jan.2018.

[10] 今金健太郎, 金井謙治, 甲藤二郎, 津田俊隆, 中里秀則, “エッジクラウドにおけるマルチメディアサービスファンクションチェイニングを活用した処理低遅延化に関する検討,” 情報処理学会 オーディオビジュアル複合処理(AVM)研究会, 2018 年 3 月 (投稿予定).